A Framework for Quality of Service over a Shared Resource and Shared Network Environment

by

Paul S. D. Card

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering
Faculty of Graduate Studies
University of Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

A FRAMEWORK FOR QUALITY OF SERVICE OVER A SHARED RESOURCE AND SHARED NETWORK ENVIRONMENT

BY

Paul S.D. Card

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Doctor of Philosophy

Abstract

"Volunteer" and Peer-to-Peer (p2p) computing are classes of resource management systems that make use of volunteered resources for off-line computation or file distribution. We propose a framework that uses these resources as a medium for deploying on-line on-demand Internet services. For our experiments we chose Web Delivery and Data Base Management System (DBMS) as together they are used to deliver wide-area Web based applications or services. They also both require computing resources and network resources. Using PlanetLab we examine several contributing factors to service performance such as resource configuration, resource loading and the wide-area network. From these results we continue to develop resource selection algorithms and a probe based monitoring system that contribute to an overall resource management framework.

COMMITTEE SIGNATURE PAGE

This dissertation was presented

by

Paul S. D. Card

It was defended on

December 19, 2006

and approved by

(Signature) _____

Committee Chairperson

Prof. Muthucumaru Maheswaran

(Signature) _____

Committee Member

Prof. Robert McLeod

(Signature) _____

Committee Member

Prof. Ken Ferens

(Signature) _____

Committee Member

Prof. Peter Graham

(Signature) _____

Committee Member

Prof. Ioanis Nikolaidis, (University of Alberta)

Dedication

Dedicated to the people that had to put up with me during the completion of this mighty tome of learning.

Acknowledgements

I would like to express my gratitude to my supervisors Prof. M. Maheswaran and Prof. B. McLeod as well as my thesis committee members Prof. K. Ferens, Prof. D. McNeill and Prof. P. Graham for their valuable suggestions, comments and for evaluating this work. I am very appreciative to TRLabs and their sponsors whose support and input made this work possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many computing applications benefit from being distributed over a local or wide area network. Often this benefit comes from centralizing data or recruiting a large number of resources to work on a problem. Resource management is a problem that has received a large amount of study since the birth of computing networks. Recently Volunteer, Peer-to-Peer (p2p), Content Delivery Networks (CDNs) and Grid Resource Management Systems (RMSs) have shown that much progress has been made for various applications. In this work, we develop a novel framework for resource management of remote resources. The framework attempts to be generic enough to allow a large variety of applications to be deployed over it. We examine some potential applications in the Literature Survey and provide conclusions. To drive the framework experiments we developed, we selected Web service as a wide area application on which to focus our work.

1.1 Terminology

We define some of the components of Resource Management System (RMSs) to standardize the language for this dissertation:

- Client Computer: The computer at a particular network location that requests and receives

the results of the application's execution.

- Application: The application is the software that will produce the results needed by the client computer. The application is the software that will be deployed onto a remote resource under direction of the Resource Management System (RMS).

- Resource: The computers that will do the processing needed to return results to the client computer, as well as the interconnecting network.

- Resource Management System (RMS): The RMS is the collection of software components that implement the control logic for placing the applications onto a particular resource for execution. It also monitors progress and takes corrective action when required results are not being achieved. The RMS may be implemented in a centralized or decentralized fashion.

These four components form the basic building blocks of Resource Managements Systems (RMS) whether it be a Volunteer Computing System, Content Delivery Network (CDN) or a Computational Grid. We will use these basic building blocks to examine existing systems in the next chapter and as a basis for our study the framework developed in this thesis.

1.2 Motivation

During the past efforts have been made to design and build RMSs that can harness the combined power of many resources. The short coming of these systems is that the perceived performance, as observed at the client, is not taken into account. The performance an end user sees is a by-product of the RMS design rather then a direct input. In effect, there is no feed back loop from the client to the RMS itself. Running an network application across various donated or best effort resources in an becomes open-loop, and forces the application to try to build in QoS without overtop of the RMS's built in decision process.

In these traditional deployments, the end-user performance can vary due to the condition of the resources being used. The interconnecting network can span 10s of different countries operated by 100s of different organizations. The remote resources can be congested, miss configured, or removed from service at any time. To this point the focus has been on building RMSs from the ground up. By building systems from the ground up, the focus is always on the low level details and the end user application is ignored. Resources or interconnecting networks for one application may not be relevant to another. For example, the bandwidth and loading of a distant resource may not be of importance if the application is streaming of a low bandwidth signal where little bandwidth or processing is needed. If the application is a coordination server for an online game application the latency may be the most relevant factor. In off-line computation the bandwidth, latency and congestion of the interconnecting network may be irrelevant.

1.3 Objectives

The development of an application focused RMS has some interesting implications.In web delivery farms that are backed by DBMSs can leverage our RMSs a web server front-end could tune its private RMS to maximum DBMS performance for itself. The web farm client could make use of the RMS to select the web servers delivering the best web performance. In other existing RMSs there is no way to tune of adjust the RMSs behavior for a given application without major reconfiguration of the system.

We intend to show in this thesis that traditional low layer measurements do not coordinate well with application performance when deployed over wide area networks of shared resources. We will show that the application itself should drive the selection of resources that take part in a deployment. We develop some skeletal algorithms for resource selection, and performance prediction, then present their performance results.

1.4 Scope

This thesis documents, the development of a skeletal RMS that places the end user application at the top of the system hierarchy. We propose that, all system performance measurements and algorithms should be driven and customized by the application contacting the RMS for service. To this end, we propose that applications should provide an application specific software plug-in that directs the RMS to evaluate and choose resources most suitable for the application driving the RMS. To reduce the complexity of this endeavor to manageable scale, the RMS we develop will focus primarily on web delivery. The consideration of other applications will not be explored.

We do not intend to explore the implications of robustness, scalability, uptime, security, trust or any of the other factors that would obviously be needed in a wide spread deployment. We do a cursory exploration at the end of these to explore what may need to addressed to further our work. For the purposes of this thesis we focus our attention on the novel aspect of the proposed system, which is to turn RMS performance focus to the application. Without first exploring this vital step, exploration of the other, equally important aspects, would premature.

1.5 Thesis Outline

In this thesis, we explore a skeletal framework for allowing remote applications to be shared or load balanced over a wide area network backbone. In Chapter 3, we examine several existing Resource Management Systems (RMSs) to determine their strengths and weaknesses. In Chapter 4, we continue to examine the impact the wide area network and the selected resources have on application performance using a PlanetLab test bed. In this examination, we test the performance of the Apache HTTP server and the MySQL DataBase Management System (DBMS) over a wide area network. From these observations, we develop a simple performance prediction algorithm and develop a resource selection algorithm using an emulator we created, Chapter 5. We discover,

in Chapter 5, that clustering resources together yields great benefit to application performance. In Chapter 6, we do a preliminary study of resource clustering. In Chapter 7, we further our clustering work by developing a distributed programmatic algorithm that clusters clients together. Finally, in Chapter 8, we revisit our emulator from Chapter 4 to evaluate the performance of the system. We then conclude and discuss future work.

Chapter 2

Literature Survey

Many computing problems can benefit from deployment on multiple computing resources. The benefits are usually better resource utilization through load balancing and better application performance due to the availability of a larger resource pool. The larger the resource pool and the larger the application set, the better the chance of finding a good balance between resource utilization and application performance. Resource Management Systems (RMSs) try to find the best resource(s) on which to deploy a given application or set of applications. The RMS can be optimized for application performance or resource utilization. RMSs can be categorized by a number of architectural attributes as shown in Table 2.1. Each of the key attributes affects the RMSs goals for allocation of applications to resources.

2.1 Performance Prediction

For a RMS to deploy application(s) onto resources it needs to be able to predict the performance of the application on a particular resource. The application's performance can be affected by the type of application, the type of resource, and competing applications running on the resource, since they may be in competition for the system's resources. In some systems there may not be

RMS Type	Grid	Volunteer	P2P	CDNs
Donated Resources	no	yes	yes	no
Number of Applications	many	many	one	many web sites
Dedicated Resources	yes	no	no	shared with QoS
Predictable Performance	yes	no	no	no, wide area affects
Level of Control	partial	centralized	fully	partial

Table 2.1: Resource management system attributes.

any competing resources and the applications may have very predictable run times. In others, the resources may be heavily loaded with applications that do not have deterministic run times. The application placement algorithm that makes use of the performance predictions can also vary in complexity and goals. For example, the RMS may be maximizing application performance or maximizing global system throughput. In the following sections we examine some existing RMSs.

2.2 Web Services

The World Wide Web (WWW) is becoming a critical component of many businesses. At one time the WWW was only a publishing system for static content. Today, the WWW includes a multitude of interactive dynamic content. Technologies such as XML-RPC (XML Remote Procedure Call) and AJAX (Asynchronous JavaScript and XML) are allowing developers to create WWW applications that react like local applications by side stepping the need for a new TCP connection and web page reloading for each interaction, as is common in traditional WWW interactions. This new found flexibility in web technology means that traditionally client based applications, such as email, calendaring and even spreadsheets and word processing, are being deployed through a web browser. Another advantage of these standard technologies is that they allow companies to

expose their APIs (Application Programming Interfaces) to the web. Today, multiple web services are being combined together to form new WWW applications often called mashups.

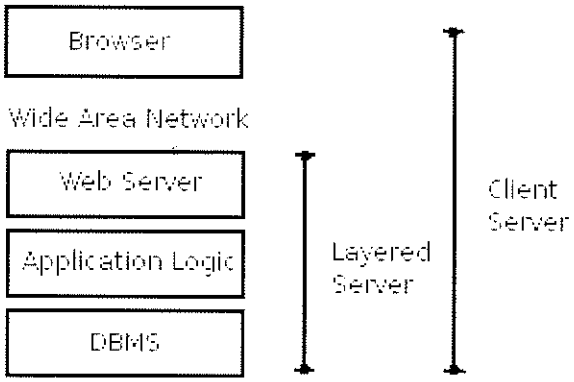


Figure 2.1: Web services architecture.

These Web applications or services are usually based on a 3 layer server architecture shown in Figure 2.1. The client connects through the wide area network to communicate with the web server. The web server in turn executes the application specific logic to generate the requested content. The application logic can be implemented in any programming language but there is a small subset of popular languages such as PHP, ASP, Ruby, Perl and various forms of Java. The application logic on any non trivial web application is often backed up by a DBMS (DataBase Management System) for content management and access to application data. For smaller web applications or services, the web server, application logic, and the database engine may reside on a single physical computer. These functions can, however, also be split over several physical computers in larger deployments. In even larger deployments any of the 3 server components could be replicated multiple times to handle a large client base. When these components are replicated and placed on separate computers they usually communicate through a shared network connection forming a WWW serving farm. The incoming requests are then load balanced over the computers in the server farm. One shortcoming of this architecture is that the number of serving computers

must be sized for the maximum load expected. Maintaining enough capacity for infrequent full loading is wasteful in terms of capital cost, power consumption and IT resources. Great savings are possible if the resources can be dynamically sized for the current workload.

Distributed Web Services

Traditional WWW services often use a simple client / server architecture. Today, more and more applications are a mashup of many existing online WWW servers. These mashups result in a single web service composed of several servers at different remote locations. Mashups show that the Internet backbone is capable of delivering services from multiple physical locations as a new combination of existing services. These ad hoc web service combinations are an example of how geographical proximity is not a requirement for successful web service applications.

From these new mashup web services, we see that even in the absence of CDNs, three layer web services can be spread across multiple underlying physical resources. Further in a mashup these resources may be spread geographically. Mashups show us that the physical location of a web service component may not be important. We explore this idea further in Chapter 3.

2.2.1 Content Delivery Networks

Content Delivery Networks (CDNs) [33] are an extension of web services. A schematic of a typical CDN architecture can be seen in Figure 2.2. To offload web farm loading, a CDN module on the web farm server rewrites the base Hypertext Markup Language (HTML) to link to content hosted on the remote CDN resources. The CDN periodically transfers a copy of the entire web farm content to its resources to keep current. One drawback of this system is the continued reliance on the originating web farm. During an originating server farm failure, the CDN cannot continue to deliver content without the page rewriting module on the hosting farm server. CDNs, however, also have many advantages for example, they can share their resources a large client base increasing

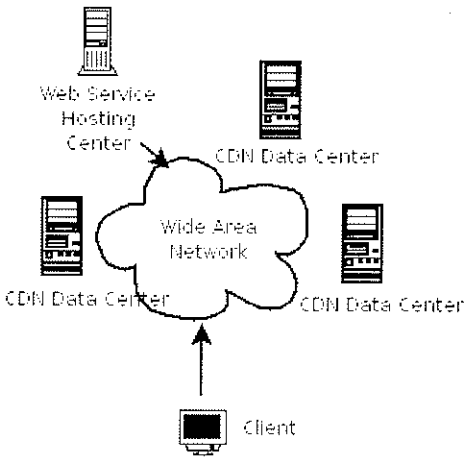


Figure 2.2: CDN architecture.

global resource utilization and thus lowering costs. Another advantage CDNs have is that they typically have many global locations allowing for the bulk of the web service to be delivered from the nearest CDN center, possibly shortening the path taken from resource to client. In CDNs the RMS is usually built in two parts. The first half is in the HTML page rewriting module that offloads traffic to the CDN content servers during periods of high origin resource load. The other half of the resource management function is at the CDN data center where the CDN needs to manage the amount of traffic it handles for each of its customers to ensure they are meeting their Quality of Service (QoS) agreements [53] [44]. The CDN data centers are usually large and have a diverse client group. This allows for extra capacity to be given to one loaded client when not being used by another. Other than meeting the QoS agreements between the CDN and the originating web service, a CDN does not need to address the special security concerns that a Grid would. As well, since CDNs only deploy one application, i.e., web services, they need not worry about the complex configurations that Grids host.

2.2.2 Multi-Computer RMS

Multi-Computer management toolkits RMS such as ROCKS [42] or Beowulf [12] are used to manage a single cluster of computers attached to a high speed local network. These management toolkits are a subclass of an RMS. In these RMSs a head node typically manages the jobs submitted to the cluster, then dispatches the incoming jobs to the pool of computing resources. In these systems the resource pool is under direct control of the cluster's head node. This makes the resource load easy to know and predict. These systems are typically accessed through a cluster Application Programming Interface (API) where the programmer can select the placement algorithm the cluster head node will choose [37]. Normally, the placement algorithm simply attempts to balance the load across the pool nodes. The RMS's goal is to virtualize the resource cluster to simplify programming and management. As the entire cluster is usually owned and operated by a single owner, setting policy and policing user loading is usually a simple task.

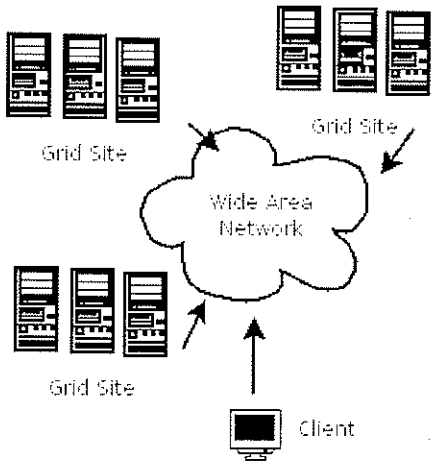


Figure 2.3: Grid architecture.

2.2.3 Grid Based RMSs

Grid systems such as Globus [21], [15] and [22] are composed of several multi-computer systems (see Figure 2.3). In grids, a grid overlay manages the cluster head nodes. The grid acts a large scale virtualization of the underlying pools of resources. Grids are typically composed of resources owned by multiple organizations and connected over a wide-area network. As the underlying resources are not as tightly coupled with the Grid overlay resource, loading is somewhat more complicated and requires tighter load monitoring. The setting of resource usage policy can be done by each member site. Large scale implementations such as Enabling Grids for E-sciencE (EGEE) [24] make use of Globus for system management and Condor [48] for workload management. One of the hurdles Grids need to overcome is wide area management. Many of the applications that run on Grids will need reliable long lived resources with security assured on the node and on the wide area network. Often applications will require complex configurations with bandwidth guarantees between subsets of resources. Grids attempt to achieve all of these goals.

The typical applications that are deployed on grids and multi-computer RMSs are high performance tasks such as:

- Distributed Ray Tracing for 3D graphics rendering [17]
- Climate Prediction [30] or [11]
- Earthquake Simulation [47]
- Nuclear Fusion Simulation [26]
- Astrophysics Simulation [41]

These application require multiple computers to achieve the volume of processing needed. They also usually have some amount of interprocess communication, timing or security concerns that do not allow them to run in Volunteer RMSs.

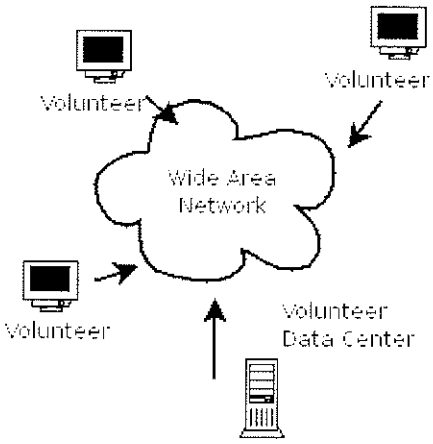


Figure 2.4: Volunteer architecture.

2.2.4 Volunteer RMS

Volunteer systems such as BONIC [7], [4] or [5] are centralized RMSs where applications are submitted to a central authority that manages a globally dispersed pool of volunteer resources (see Figure 2.4). Which are most often dedicated to some primary (i.e. non-RMS) task. When idle, the resources switch to the application provided by the central RMS. Application performance prediction becomes very hard in this environment due to the inability of the RMS to predict when a resource will be dedicated to the RMS or to its primary task. To do performance prediction these system use a large aggregation of the total system capacity splitting the application across hundreds of machines. With the large variety of resources and network connections in use inter-application communication is not practical. Volunteer systems typically only support off-line style processing with no communication between resources. This is a strong limitation of the type of application that can run on Volunteer RMSs. The class of problems best suited for this type of RMS are referred to as "Embarrassingly Parallel" for their inherent suitability to processing by many independent resources. Volunteer systems generally do not provide any security measures

on the resource or network connections. They often need to schedule multiple executions of each application to allow for consensus checks to ensure that the results have not been tampered with by the volunteered resource.

Some examples of application that make use of Volunteer Computing RMSs such as, BONIC are:

- SETI@Home: Searching radio telescope data for extra-terrestrial signals [3]
- Rosetta@Home: Finding the 3 Dimensional shapes of proteins [1]
- Seasonal Attribution Project: Global weather simulation [2]

These applications do no have any interprocess communication. They also can ensure quality of results by having calculations repeated on many hosts and comparing results. The data security in these applications is not sensitive in nature.

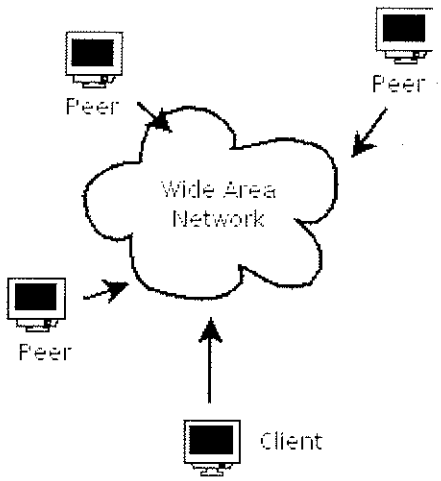


Figure 2.5: P2P architecture.

2.2.5 Peer-to-Peer (P2P)

Peer to Peer (P2P) systems (see Figure 2.5) are designed from the ground up for a single application, often file distribution [46]. P2P system rely on users donating resources of network bandwidth and storage to the system in a tit-for-tat manner where users gain from the RMS in proportion to what they contribute. As early P2P systems were affected by legal difficulties over the content rights management, new P2P system are fully distributed. The fully distributed nature makes them very robust to network fragmentation or individual systems failure. By limiting themselves to a single application, p2p systems can better optimize their performance. This could include building in additional security over what can be provided by a Volunteer RMS. One drawback of p2p file distribution systems is the digital rights that can be violated. Because of the abuses that often happen on p2p systems, their reliability is often uncertain. Despite an evolution towards fully distributed algorithms (at the cost of performance) legal exchanges still surround most p2p systems making them unreliable for long term availability. Being application specific limits their application to high performance computing. They do have the potential to evolve into a general purpose RMS.

2.3 Motivation

Our proposed framework maintains strengths from the above systems while attempting to suppress their shortcomings. In our framework we propose to make use of volunteer resources as is done in volunteer computing. Rather than simply supporting offline style processing, however, we propose to support inter-resource communication as is done in grid systems and cluster computing. For the moment we do ignore the security and trust considerations that grids implement to simplify our design. These items can be re-introduced at a later time using existing trust models and encryption as done in existing RMSs. We see web services such as those deployed over CDNs as our initial

application. We will not sacrifice support for offline processing or large computation problems that grids support. We would also like to support some amount of QoS as is done in CDNs. We would also like our framework to be decentralized and robust as p2p networks are by using decentralized algorithms [46].

We expect the resulting system to reduce the computing capacity needed by individual applications by being able to use resources from our framework in a CDN style. We also expect our framework to increase global resource utilization by allowing idle resources to contribute to the framework. We expect the resulting framework to be a component of a larger Public Computing Utility (PCU) described by Maheswaran et al. [29].

Chapter 3

Framework Architecture

One of the problems with attempting to harness shared resources over shared networks is the associated *quality of service* (QoS). Some mechanism is required to ensure that the resources promised will be delivered; otherwise, the job should not be accepted [56] [39]. To further complicate things, some online applications need to measure how successful or "good" the delivery was. For example, it is not enough in a streamed movie to deliver it to the customer; the jitter and burstiness of the delivery must also be managed. If the framework is contracted to deliver a service it needs to arrive at the clients location, with a specific level of quality. This can be a very challenging problem when one considers that the resources employed to deliver the service can switch back to their primary use at any time, as is the case for shared resources. In addition, the interconnecting network being used to deliver the service can become saturated with competing traffic.

We have developed a new framework for clients to find on demand resources for online applications. In our system we propose to start with the end user's location, and from there allow them to specify what QoS is required for each application. That means that the service being contracted is always measured in terms of delivered performance to the client at his or her location. The measurement and evaluation occur at the end host itself or, when more appropriate, at a gateway device near the client. We also include the caveat that the performance is measured by the applica-

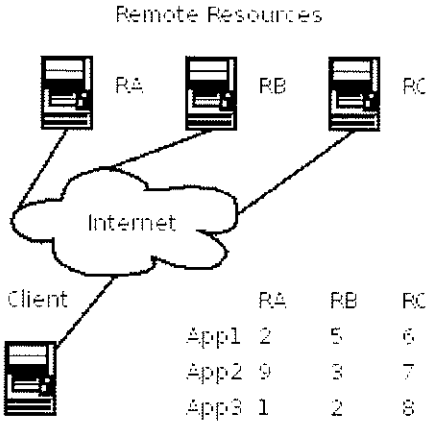


Figure 3.1: Proposed system.

tion software. This implies that, for each application or dataset, we require an application-specific plug-in to return an assessed performance value. This plug-in may also go one step further and be dataset specific where the dataset would drastically alter application performance. This means that the set of resources needed to satisfy a request or demand will be different for each application, as the plug-in may have different measures of QoS requirements. In Figure 3.1 we provide an example of a table of remote resources and their performance for different applications. From such a table the local anchor point would then decide on a set of resources on which to deploy the applications. Previous work in such tunable applications has to the best of my knowledge only been done in the context of mobile computing [20].

Doing performance evaluation at the client has had some indirect study. Rolia et al. [38] have looked into QoS in resource pools. Cherkasova et al. [14]have studied end to end server performance looking for bottlenecks and Ruan et al. [40] have studied the network effects on providing services over a wide area. These works, together, point us towards moving to an application layer measurement of performance.

One problem we anticipate is bootstrapping new applications. To create new application plug-

ins the characterization of the client's needs can be performed by running the application. In our framework applications are profiled by running them on a random set of resources. From this trial set we generate a weight vector by comparing this performance to the existing probe data for each of the resources used. We can then use ongoing probe data to predict a baseline application performance for each application and each resource. As the application runs we can then switch to using the live returned performance data as discussed in Chapter 5.

3.1 Framework Components

Our framework consists of a number of software elements; the client software, and resources. It also includes a host near the client that acts as a local resource broker we call an "Anchor Point". These may all be deployed on a single physical resource or spread across a wide area network. In this thesis we assume that the resources and anchor points are deployed on several machines distributed over a wide area network.

3.1.1 The Client

The client in our model needs to be connected to the wide area network where the framework is deployed. It must also have the application and dataset in a packaged form such as a Linux RPM (RPM Package Manager)[10] format or equivalent. The file must be publicly available for a download manager such as Linux YUM (Yellow dog Updater, Modified)[52]. The client then makes use of an anchor point list that is posted on a central server to find nearest anchor point. It then contacts that point with the application file that is to be deployed. The anchor point will return with a weight vector that is used to generate performance predictions. Once the weight vector is generated, the anchor point will be ready to give initial performance predictions for any local client. The client can then make demands for service to its location. As more application

deployments are executed, the anchor point (see Section 3.1.2) will learn the characteristics of this application, improving their performance predictions.

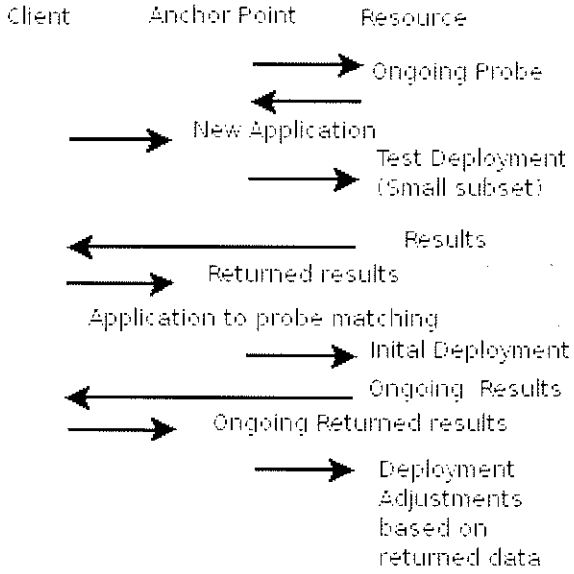


Figure 3.2: Message sequence chart.

3.1.2 The Anchor Point

The anchor points act as local brokers for the RMS, refer to the message sequence chart Figure 3.2. They maintain the current conditions and loading of remote resources. An anchor point consists of two major sub components. The first is a probe thread of execution that probes a pseudo random subset of remote resources on a predefined schedule. Each probe consisted of 10 different tests, which we discuss later in the thesis. During our testing we used a four hour schedule, and a subset size of 50 of the 500 available nodes, to maintain our baseline resource performance measures. From the baseline probe values the anchor point forms a signature that represents its position in terms of remote resource performance. The signature is a vector of remote resource performance for each probe test evaluated. The anchor point will then contact other anchor points to see if they

are available to cooperate. In order for two anchor points to cooperate they need to have the same view of remote resources. This test for cooperation is a simple comparison of probe signatures. If the signatures are within the system wide tolerances they form a cooperative cluster. The signature comparison algorithm and tolerance values are discussed in Chapter 6 and 7. As we will show later these clusters so formed tend to be geographically clustered on a city wide scale for our test environment. The anchor point clusters work cooperatively to maintain a shared database of application performance and resource loading.

The second function the anchor point performs is resource selection for application deployment. When a client first contacts an anchor point with a new application it needs to have an application weight vector generated. To generate the weight vector the anchor point deploys the application to a subset of resources (see Chapter 6). After a short test, the anchor point compares the delivered performance to its probe signatures to generate a weight vector for the application. Then when the application is initially deployed the weight vector in combination with the current resource probe data will be used for performance prediction. As performance results are returned from the deployed applications the anchor point broadcasts the results to its cooperative cluster. As the results for the cluster accumulates the performance predictions shift to our resource selection algorithm described in Chapter 5.

Similar frameworks that use anchor points, or "landmarks", exists in experimental Label Switched Paths (LSP) [34] and sensor networks [31]. In these situations the landmarks use either fixed WAN locations or they are used for host positioning. In these all cases to purpose of the "landmarks" is to find a nodes network position in terms of connectivity. Our framework places these node on the edge of the network and perform significantly different and larger task then traditional "landmarks". This is the reason we describe our "landmarks" as anchor points. Our anchor points are not low layer static positioning nodes, but application performance monitoring and application deployment agents that work on behalf of the client.

It should be noted that, the anchor points performs a local cooperative load balancing among

the cluster to help reduce the chance of swamping out nodes. There may still be interference on the resource from other, non cluster anchor points, but that should show itself when application performance suffers reducing client perceived performance. At this point the global average of performance for this resource will be reduced as will the cluster's evaluation of the node. Further discussion included in Chapter 5.

3.2 The Resources

The resources used in my experiments are Linux computers that make use of the Linux VServer virtualized sandbox. We exploit the VServer slice abstraction provided by PlanetLab [35]. The Vserver in a kernel based virtualization that provides many simulated Linux systems on a single hardware instance. The resource must then set its use policy in terms of the resources it is willing to donate to the framework. The resource then responds to application deployment requests from the framework anchor points.

The other function each resource provides is tracking the performance of applications that are currently running and those which have completed. For applications that have recently run on the resource, the resource maintains an average performance, as observed from each client location. This published average is used by anchor points in their performance predication algorithm described in Chapter 5.

3.3 Cascading Deployment

In our framework we do not explore larger more complicated deployment beyond basic client / server. In a system deployment we envision that our framework could be used in a cascading deployment style. In the case of a multi-layer web service, the head en web server could contact our framework to provide backend DBMS services. From the client perspective we would in effect

have a cascading service.

3.4 Outstanding Questions

Certain fundamental questions regarding this proposed framework still need to be answered. First, how does low layer measures correlate with application performance in a shared-resource and shared-network environment? It is possible that traditional low layer performance prediction [28] [50] [54] [27] [57] may be better suited to this task. Second, by what mechanism can one predict the performance of a resource for a specific plug-in, given only its performance history [55] [45]. Thirdly, we examine the clustering of anchor points using resource performance probes to look for system performance gains. Finally, one must be able to decide which resource to select given a set of demands with their predicted performances. The rest of this thesis describes our examination of these questions and the framework components.

Chapter 4

Application Performance vs. Layer 2, 3 and Host Performance Measurements

We propose, that in a shared network/resource environment, traditional low-layer performance techniques [49], such as measuring of network or resource load [54], are not appropriate for determining the performance of an application or service. To test support this statement, we designed a number of experiments using the PlanetLab network. PlanetLab is a network of globally-distributed resources. These resources are donated by participating institutions, along with the associated network bandwidth, in return for access to a virtual slice of all of the participating nodes. These nodes and their networks tend to be loaded by researchers running network experiments. As the nodes are virtualized, we have no way of knowing the exact nature of the load or the number of participants. This makes for an excellent test bed for our purposes. Without knowledge of the loading on the network and resources, we can test the low-layer techniques in a real-world setting. We believe that, if our techniques show promise in the unpredictable environment of PlanetLab [35], where we cannot predict what other processes are running, they should perform even better in a SETI@Home style network [6], where machines that when idle are dedicated to the Resource Management Framework (RMS). In these cases the RMS would be the only one submitting jobs

to an otherwise idle machine.

4.1 Application Performance vs. Layer 2, 3 and Host Performance Measurements: Experiments

To evaluate the performance of traditional host and layer 3 measurements vs. application performance measurement, we pseudo randomly selected 25 nodes from the PlanetLab network. Each node was configured with the following software:

- Apache HTTP Web Server. We wanted to include a real world dataset for our tests. However, because the size of standard datasets is prohibitive, we designed a 15MB workload that had file size and request frequencies to match the well-known University of Saskatchewan dataset [9]. This allowed us to have a workload with real-world characteristics, without needing to use huge sets of web server traces. The total number of requests in our workload was 5000 documents.

- Apache Flood. Apache Flood is part of the Apache family of software designed for load-testing web servers. We used this program to load the remote server, and to generate a performance evaluation of the Web Server, based on the time in milliseconds it took to download the workload. The Apache Flood program creates a separate TCP connections for each download.

- MySQL *Data Base Management System* (DBMS). MySQL is an open source database engine. We employed it with 3 different workloads. Each was designed to progressively increase the load on the server and network.

  - Workload 1: This consisted of an SQL query that generated light server load and light reply volume. The query is a 2-table join which results in a complete data set of about

50,000 rows. We limited the return set using the `LIMIT` SQL option. The database is part of the standard MySQL test library. By limiting the result set to 1000 rows, we reduce the returned data volume and the work load. By using `LIMIT`, the DBMS query processor stops the table join once it has reached the limit value, thereby reducing the workload.

  - Workload 2: This query generated a larger server load and a larger reply volume. The only difference between this query and the first is that we increased the limit by a factor of 10, setting it to 10,000 records.

  - Workload 3: This query had a very high reply volume and high server load. In this final query we did not limit the return size of the query, letting the DBMS process the full join and return the 52,000 record result set.

- Ping Time in Milliseconds, measured using the system `/bin/ping` program from client to server. The packet size used was 56 bytes, which translates into 64 ICMP data bytes when combined with the 8 bytes of ICMP header data. This measure returns the latency of the interconnecting network.

- Server CPU Load Average over the past 15 minutes, as reported by the `/proc` file system (kernel interface). This returns the current loading of the server resource.

- Installed Server CPU clock speed, as returned by the `/proc` file system. This provides a good indication of the server's unloaded performance potential.

- Server Installed Memory, as reported by `/proc` file system. This is another good indicator of the computer's unloaded performance potential.

- Distance from server to client in Km, measured as the crow flies. Each PlanetLab node has a latitude and longitude entry in the PlanetLab database. We used these values to calculate

the geographical distance between nodes. This was included as an additional measure for its convenience. We do not expect it to correlate at all with server performance.

- We also installed a series of benchmarking programs to test the servers' live performance near the time we tested it with the other benchmarks. We wrote these ourselves to bypass the need for remote Secure Shell (SSH) connections needed to deploy standard benchmarking utilities. Writing our own tests allowed us finer grain timing. It should also be noted that we are trying to test application interfaces to the application-layer. By writing our own benchmarking tools we test exactly what an application developer would see rather then the underlaying hardware that most available benchmarking tools try to exploit. As an example we are interested in how fast we can load a web workload in to memory, not the raw disk speed available. We belive that our simple benchmark tools are more appropriate, for our purposes, then available benchmark tools that look for raw hardware performance.

  - TCP test program, used to deliver a test stream of TCP traffic from the server to the client. This was a small Java client / server pair that moved 5MB of data between the 2 nodes. We wrote this program, rather then use an existing benchmark, to gain tighter control of the remote timing of the test. We open a second set of TCP ports on each of the client / server pair to allow signaling of test start / stop and timing. Using traditional benchmarks would have meant starting and stopping tests using an Secure Shell (SSH) connection, a much longer process. Typical run times were from 0.2 to 0.7 minutes.
  - UDP test program, used to deliver a UDP stream from the server to the client. A Java client / server pair was used to move 5MB of data using the UDP protocol. We found that over 99.2% of UDP transfers, when 5MB was transfered, the full 5MB was received. We did not examine if there was any data corruption in the transfer as most UDP applications would be robust to some data corruption. Also, most single packet corruption would be detected by the UDP checksum and would appear as a lost packet.

We also did not look into packet reordering. As with the TCP benchmark, we wrote the application to gain tighter timing control. Typical run times were from 0.2 to 0.7 minutes.

- Integer Benchmark: We wrote a small program to test the performance of the server performing integer manipulation. typical run times for this program was 20-200 seconds. Again, we wrote our own simple program to tighten timing control.
- Floating Point Benchmark: To see if the resource performed any better on floating point calculations, we also wrote a floating point benchmarking program. Typical run times were from 2 to 200 seconds.
- File I/O Benchmark: As a memory-limited web server must perform many file operations, we also wrote a small file I/O test program that reads the web server workload into memory. This should be useful in determining whether the performance is file I/O limited. typical run times were from 2 - 20 seconds. This was a Java based implementation and like the other benchmarking programs not as sophisticated as other available benchmarking tools. These tools were however more then sufficient to show relative performance between nodes. We did not intend to compare the relative speed of PlanetLab nodes using benchmarks to other computers, as such there was no need for standardized benchmarks. This implementation used as standard blackdown java 1.4.2 file socket.

We pseudo randomly selected 25 nodes from the PlanetLab network, then had each of the 25 nodes test all of its peer nodes, giving 25 x (25-1) = 600 tests.

The largest amount of development time was spent creating a quick deploy/test environment for the above tests. We wanted to make sure that all of the tests happened as close to one another as possible, to avoid having other activity skew our results. Otherwise other PlanetLab researchers' actions might change the resource or network conditions. The tools I created allowed us to quickly

deploy and run multiple tests at one time. We also scheduled the tests to ensure that they would not interfere with each other. The tools allowed us to complete the 600 tests within 6 hours. As the nodes were randomly selected they would naturally be spread around the globe, avoiding time of day fluctuations.

4.2 HTTP Application-Layer Testing: Results

As expected, we found that in the shared network and resource environment of PlanetLab, the traditional low-layer measurements do not provide a satisfactory correlation with the measured application performance. We began by examining the performance reported by Apache flood, downloading our custom workload from a remote server running the Apache HTTP server. We compared the reported performance with various traditional measures.

Benchmark	MySQL 1	MySQL 2	MySQL 3	HTTP
Integer Benchmark	.07	.09	.05	.06
Floating Benchmark	.15	.10	.11	.22
File IO Benchmark	.03	.03	.00	.13
15 min Load Average	.06	.03	.02	.14
Server Memory	.13	.10	.06	.18
CPU Clock Speed	.15	.32	.12	.19
TCP Benchmark	.72	.53	.63	.69
Hop Count	.30	.29	.31	.58
Distance	.00	.39	.13	.16
Ping Latency	.53	.54	.57	.87

Table 4.1: Absolute value of the Pearson correlation of tests.

The Pearson Correlation r_{xy} is defined as:

$$r_{xy} = \frac{\sum(x_i - m(x))(y_i - m(y))}{(n-1)s_x s_y}$$

Where $m(x)$ is defined as the mean of the x sample and

s_x is defined as standard deviation of sample x

Pearson correlation equation. (4.1)

We calculated the absolute value of the Pearson correlation factor (see Equation 4.2) for each of the low level measures vs. the performance of the remote host acting as a server, summarized in Table 4.1. The absolute value of the Pearson correlation factor ranges from zero to one, where zero has no correlation and one is a linear correlation. We would like to see correlations above .75, we however observed that most results have low very little corelation. Ping latency and the TCP benchmark both reflect some correlation of the performance of the resource as a server. Unfortunately, only in the single case of ping latency vs. HTTP performance do we start to see some corelation above .75. We conclude that for a general purpose deployment, ping latency alone would not be enough to predict future resource performance. We make use of the Pearson corelation throughout the following section.

We first examined the server resources to see if this had any effect on the performance of the machine as a web server (see Figure 4.1). I have plotted the server performance as time in seconds to download the workload versus the installed CPU in MHz, as well as the amount of installed memory in MB. We found the Pearson correlation of these tests to be .18 and .19 respectively this indicates that there is no significant correlation between the machine configuration and the performance of the machine as a WWW server.

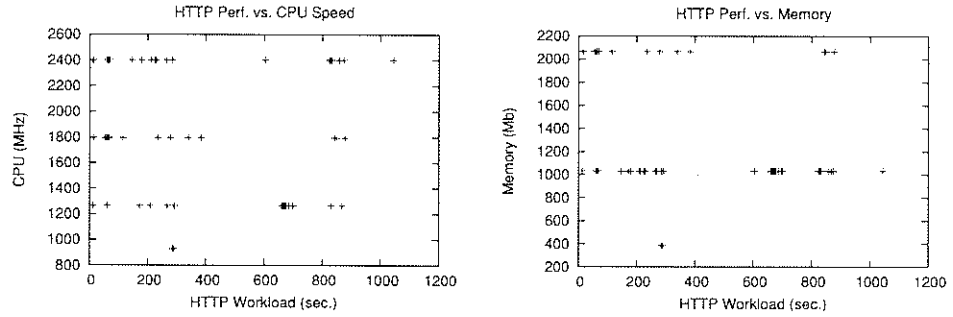


Figure 4.1: Apache performance vs. CPU speed and system memory, respectively.

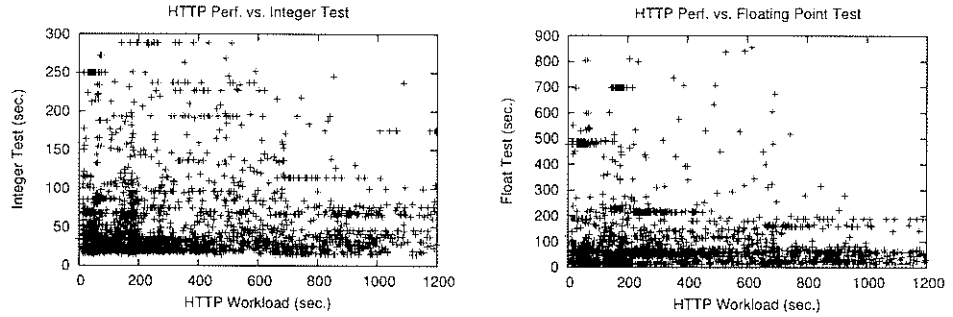


Figure 4.2: Apache performance vs. integer and floating point benchmark, respectively.

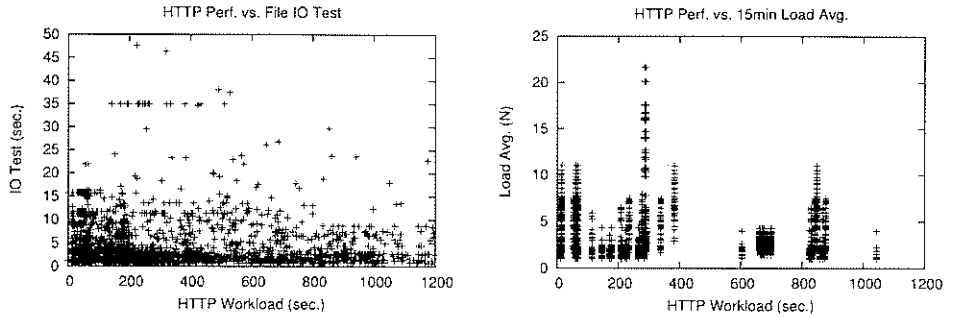


Figure 4.3: Apache performance Vs. file I/O benchmark and 15 min. load average, respectively.

We then observed the server performance running a few benchmarking routines to test the machine performance near the time we downloaded the web workload (see Figures 4.2 and 4.3). As can be seen, the raw performance of the server does not seem to influence its performance delivering our workload, nor does the reported 15 minute load average. Correlations of .13 and .14 respectively.

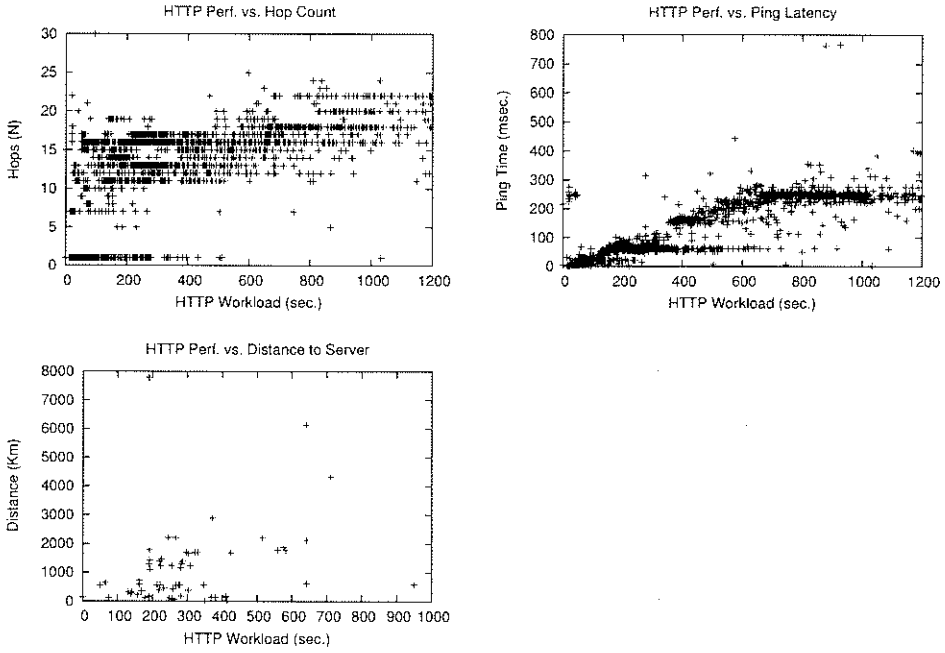


Figure 4.4: Apache performance vs. hop count, latency and distance, respectively.

As the previous resource tests did not show any strong correlation, we assumed that the bottleneck must be in the network. We examined the HTTP performance versus the number of hops, latency and distance (see Figure 4.4). Correlations of .58, .87 and .16 respectively. We observed from the plots and corelation figures, the number of hops and latency did have some correlation with the resource performance. We assume that this is due to the heavy reliance of the HTTP protocol on the TCP protocol. As TCP uses many packet exchanges to confirm data transmission and to handshake, latency would affect the server performance. This is because the initial handshake

must be repeated for each of the small file transfers. In our case there are 5000 TCP connections each one would have to wait for the SYN/ACK to occur before the transfer could being.

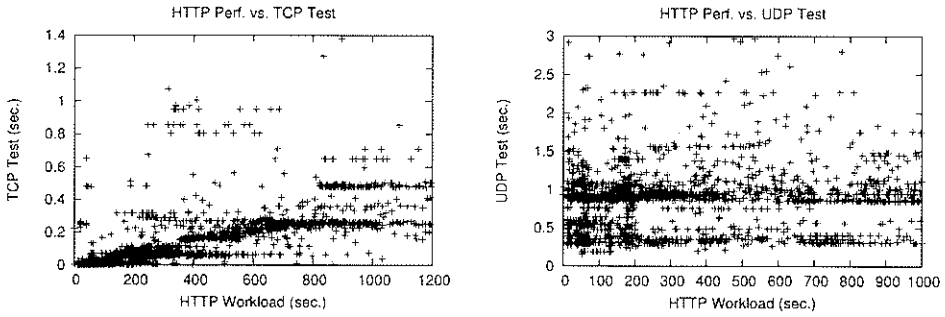


Figure 4.5: Apache performance vs. TCP and UDP benchmarking program, respectively.

Finally we compared the results from two network benchmarking programs with the HTTP performance measures taken (see Figure 4.5). As shown, the TCP performance benchmark does again exhibit some correlation (.69) with server performance. We expect that this can be explained by the same HTTP reliance on TCP as earlier. The UDP test did not correlate well (.21) with the server performance, which leads us to believe that the raw bandwidth between the server and the client is not a limiting factor in HTTP performance for me experimental parameters.

4.3 MySQL Database Engine Experiment: Results

We next observed the performance of a remote resource installed with the MySQL DBMS. We compared its performance serving a light query to a remote client, and compare the results to the same previous low-layer techniques.

We began by looking at the raw performance figures of the server vs. the performance of the resource as a MySQL server (see Figure 4.6). As one can see from the first workload, again low correlations of .13 and .15 respectively.

We then compared the other 2 MySQL workloads to determine the effect of increasing the load

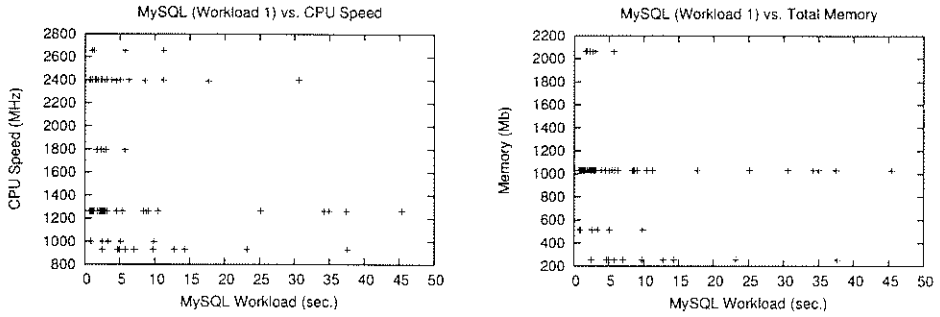


Figure 4.6: MySQL workload 1 vs. CPU speed and total memory, respectively.

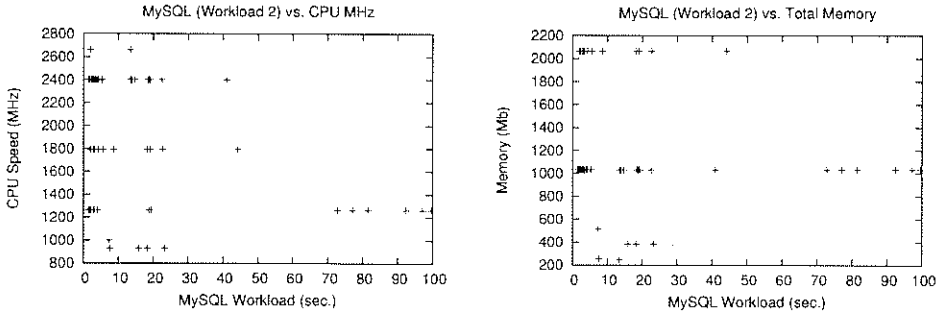


Figure 4.7: MySQL workload 2 vs. CPU speed and total memory, respectively.

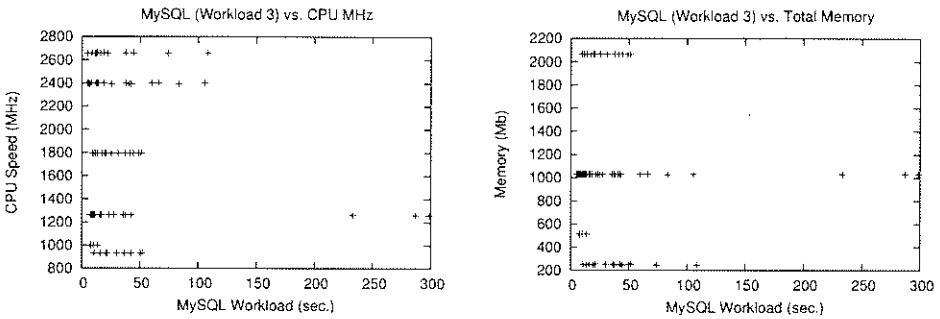


Figure 4.8: MySQL workload 3 vs. CPU speed and total memory, respectively.

(see Figures 4.7 and 4.8) We found that increasing the volume of the workload did not seem to greatly change the dependence of the server on the raw computing power, (.32 and .10) and (.12 and .06) respectively. This leads us to believe that, in a real-life deployment, the raw performance of a computer will have little effect on the performance of online applications. It will be more important in offline style applications that do not heavily rely on the network.

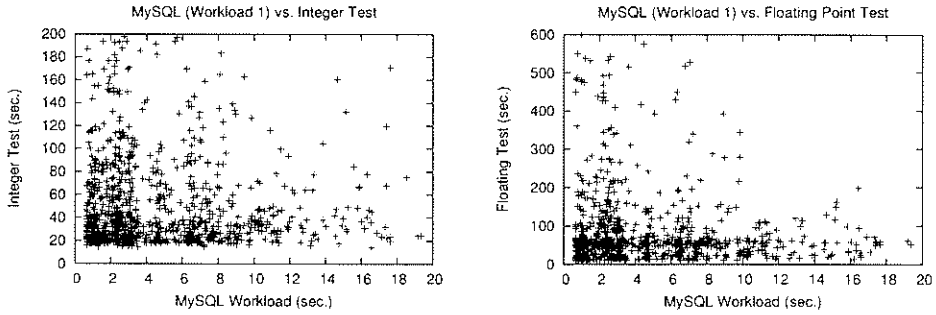


Figure 4.9: MySQL workload 1 vs. integer and floating point benchmarks, respectively.

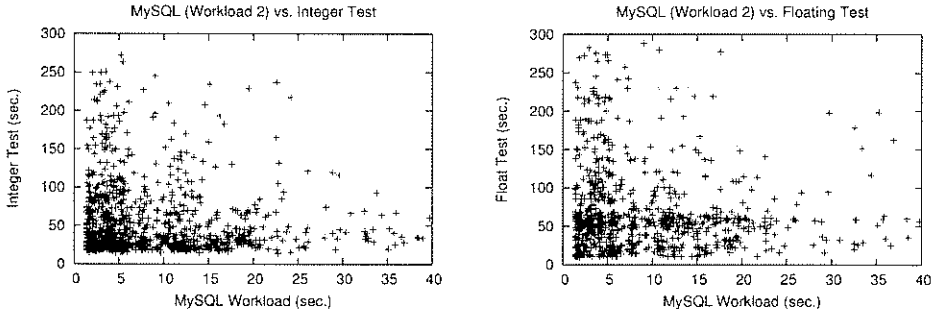


Figure 4.10: MySQL workload 2 vs. integer and floating point benchmarks, respectively.

Given that the raw unloaded performance of a machine did not affect the performance of the DBMS, we then looked into how the current loading of the machine would affect the performance that a given resource could provide. We started by comparing the performance of the 3 workloads vs. our floating point and integer benchmarking programs (see Figures 4.9, 4.10 and 4.11). We

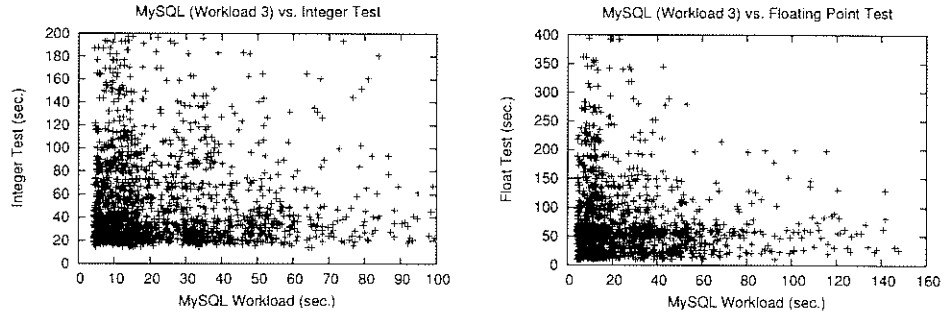


Figure 4.11: MySQL workload 3 vs. integer and floating point benchmarks, respectively.

expected that, with the 3 workloads, we should be able to see some performance variation of a machine with increased loading. However, the results suggest no such strong correlation. The correlations we saw for workload 1 were .07 and .15 for workload 2 .09 and .10 and for workload 3 .05 and .11 respectively.

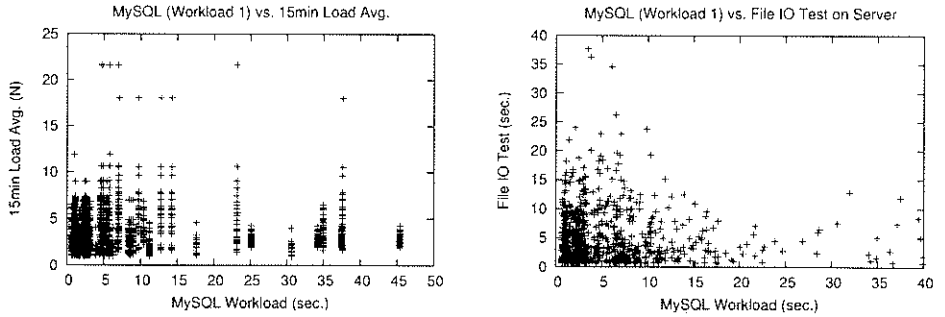


Figure 4.12: MySQL workload 1 vs. file I/O benchmark and 15 min. load average, respectively.

We then compared the performance of the 3 workloads with the 15 min load average reported by `/proc` to the DBMS performance (see Figures 4.12, 4.13 and 4.14). Correlations were .06, .03, and .02 for loading and .03, .03, and .00 respectively for file I/O performance. The plots and correlations again show that neither the load average nor our I/O test program provided good indications of the performance of a resource such as a MySQL server.

As with the HTTP performance tests, we observed very little correlation of the resources'

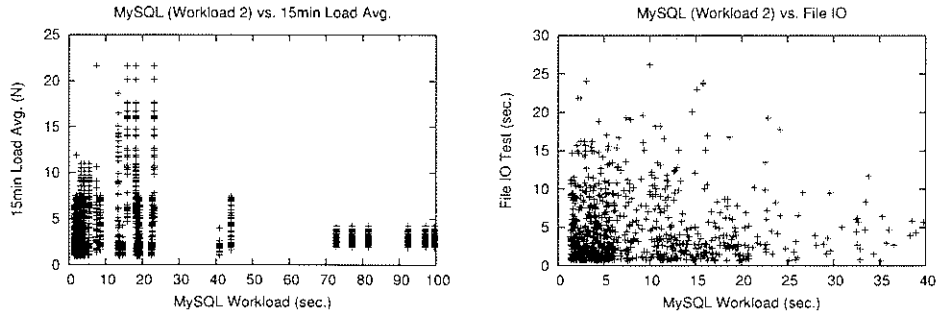


Figure 4.13: MySQL workload 2 vs. file I/O benchmark and 15 min. load average respectively.

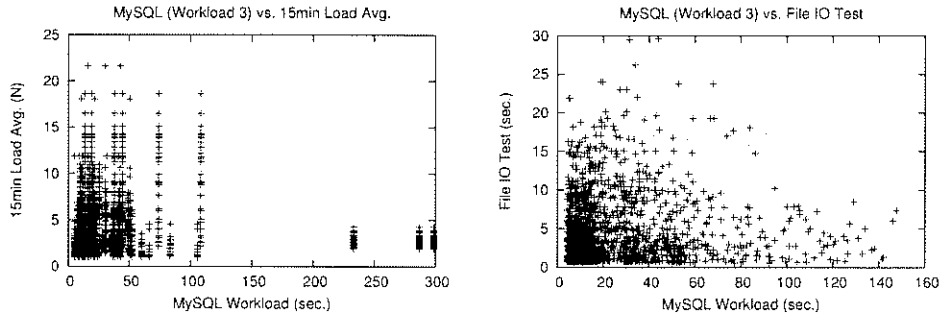


Figure 4.14: MySQL workload 3 vs. file I/O benchmark and 15 min. load average, respectively.

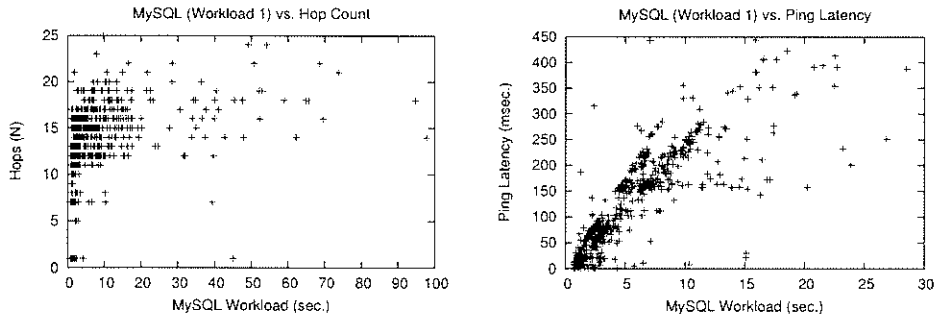


Figure 4.15: MySQL workload 1 vs. hop count and latency, respectively.

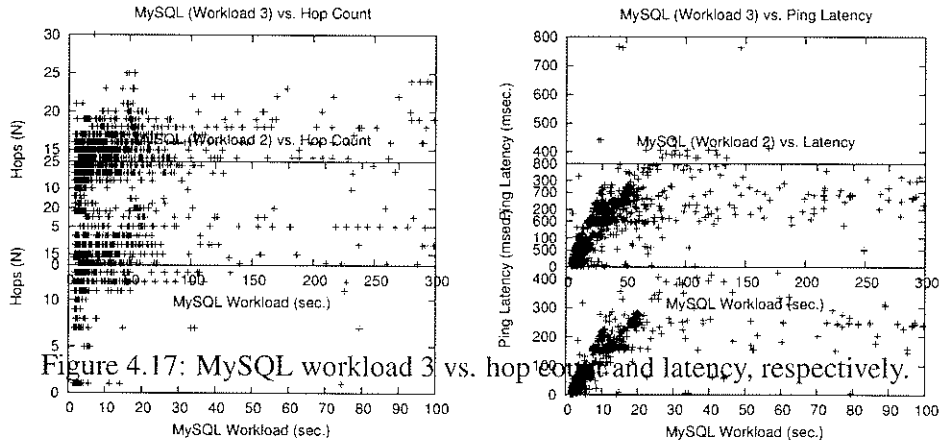


Figure 4.17: MySQL workload 3 vs. hop count and latency, respectively.

Figure 4.16: MySQL workload 2 vs. hop count and latency, respectively.

performance to their performance as a DBMS server. We then examined the network for a stronger influence on performance. We began by looking at the hop count and latency vs. the DBMS performance tests (see Figures 4.15, 4.16 and 4.17). Correlations were .30, .29, and .31 respectively for number of hops and .53, .54, and .57 respectively for latency. Examining the results, we see little correlation between the latency and the resource performance.

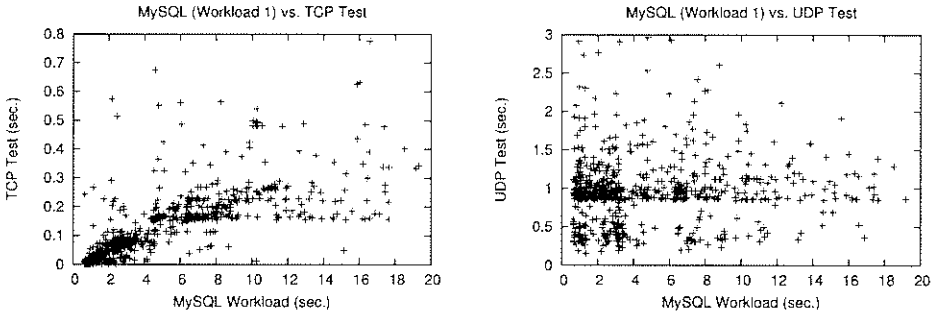


Figure 4.18: MySQL workload 1 vs. TCP and UDP benchmarks, respectively.

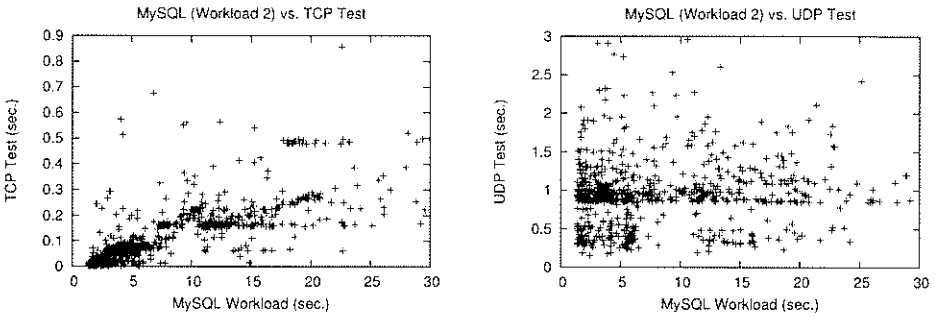


Figure 4.19: MySQL workload 2 vs. TCP and UDP benchmarks, respectively.

To verify the results and to complete the tests we looked at MySQL performance vs. our TCP and UDP benchmarking tests (see Figures 4.18, 4.19 and 4.20). The correlations were .72, .53, and .63 respectively for our TCP benchmarks and .29, .28, and .12 respectively for our UDP benchmark. Again, There is little correlation. This is unlike the HTTP test that did show some

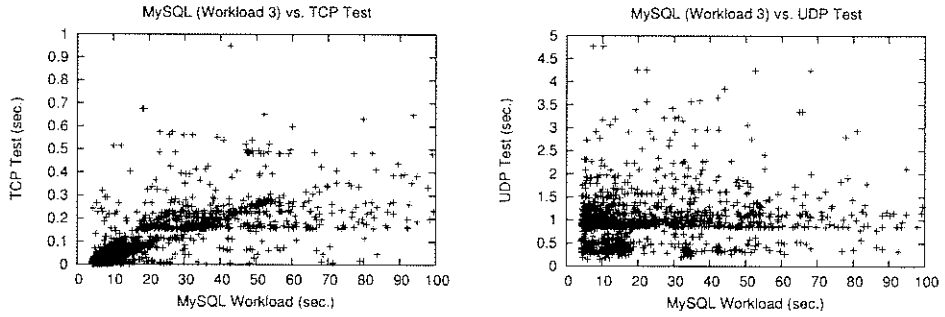


Figure 4.20: MySQL workload 3 vs. TCP and UDP benchmarks, respectively.

correlation. We believe that this may be due to the fact that the HTTP workload was larger than the result set returned by the DBMS test.

From the above results, we further conclude that traditional low-layer performance measures have limited value in the shared network/resource environment of PlanetLab. We therefore choose to explore other mechanisms to predict performance. If we want to consider a range of other applications, we believe that the only general way to unify the measuring of remote application performance is to do so by using only application performance.

4.4 Noteworthy Results

One of the interesting results we found was that the performance of a resource varies drastically, depending on where in the network one measures its performance. We found that a resource's performance can vary by as much as an order of magnitude depending on where the measurements are taken. We also noticed that the variation in the evaluation of the remote resource shrinks as the resource measurement points approach each other in the network. Closer neighbor clients tend to have similar evaluations of the performance, whereas two distant peers are likely to have very dissimilar evaluations of a remote resource. To verify these findings we pseudo randomly selected 10 remote servers from PlanetLab, and plotted the difference between a server evaluation

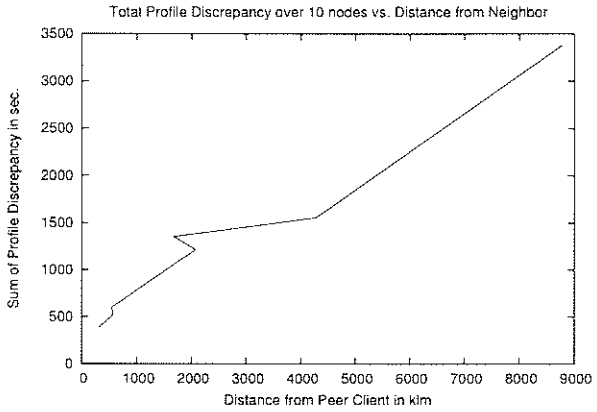


Figure 4.21: Difference between peer client evaluations vs. distance between peer clients.

taken by a 2 clients at different locations vs. the distance between the client nodes evaluating the server (see Figure 4.21). One observes almost a linear relationship. We belive that this is due to the network influence. The closer two evaluation points are, the larger the number of common network elements. In retrospect this should be an intuitive result.

To further verify these findings, we plotted the performance of a resource as a server evaluated by a peer node at the the same site vs. the evaluation done by a peer at a remote site (see Figure 4.22). If there was little effect on the network position we would see a strong correlation between the 2 measures and a strong linear relationship. When we calculated the Pearson correlation we found a value of 0.12, reflecting little correlation. One of the effects that can be seen in the plot is the two strong horizontal lines through the plot. In this plot two sizes of test can be seen. The first one takes about 200 seconds to complete, the other 20 seconds, when evaluated by a peer at the same site. When the same resource is evaluated remotely, its value varies wildly over the x axis.

From these results we see that a server's value is highly dependent on its location in the network. One of the drawbacks of evaluating server performance at the client is scalability. As we have seen, the performance is highly position dependent.

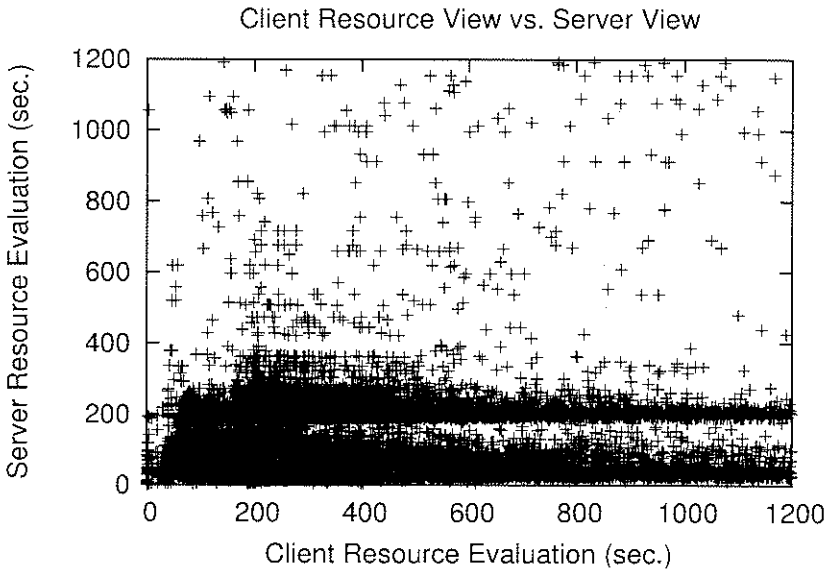


Figure 4.22: Server performance evaluation from client next to the server vs. remote client server performance evaluation.

4.5 Testing Conclusions

As shown from our first set of experiments, traditional measurement techniques do not correlate well with actual server performance in a shared network and shared resource environment. We also determined that a client's evaluation of the network resources is highly location-dependent. Nearby clients tend to have a very similar view of the network. This suggests that nearby nodes could cooperate in building a shared network view.

Chapter 5

Framework Components

In this chapter we explore some components we used to build our framework.

5.1 Performance Predictions

From our first set of tests we concluded that one should not look into the low network layers to do remote resource performance predictions. Our goal is to predict the future performance of a node based solely upon its past performance at the application layer. Similar work has been done in other domains [45] [36] [16] [58]. If this prediction is possible, it could potentially provide much greater value than the traditional measures, since the high-level measurements would, in effect, encompass all of the low-layer details. Unfortunately, it may be that predictions would become specific to the application and remote resource specific. However, the predictions could be shared amongst a pool of clients that had the same evaluation of the remote resource.

5.1.1 Performance Predictions Experiments

To test our theory that the future performance of a remote resource can be predicted, we simplified our previous test suite. As many of the results are the same for the MySQL workloads and the

Apache workloads, we eliminated the MySQL tests as well as the low-layer testing. As the testing will be long term, the reduction of testing volume helps us remain good PlanetLab citizens. We believe that the Apache server and its workload is a close match to a real-world application since the workload is modeled on real data while the MySQL workload was simply a convenient test.

We randomly selected 50 nodes from the PlanetLab network for our tests. Each of the selected nodes, on a 4 hour cycle, would test all of its peers' performance with our workload. One of the drawbacks of PlanetLab is that it is a test network. We found that it requires 50 nodes to keep approximately 35 nodes configured and running. Our tests executed for approximately 8 months, generating about 300 tests an hour.

5.1.2 Results of Performance Prediction Experiments

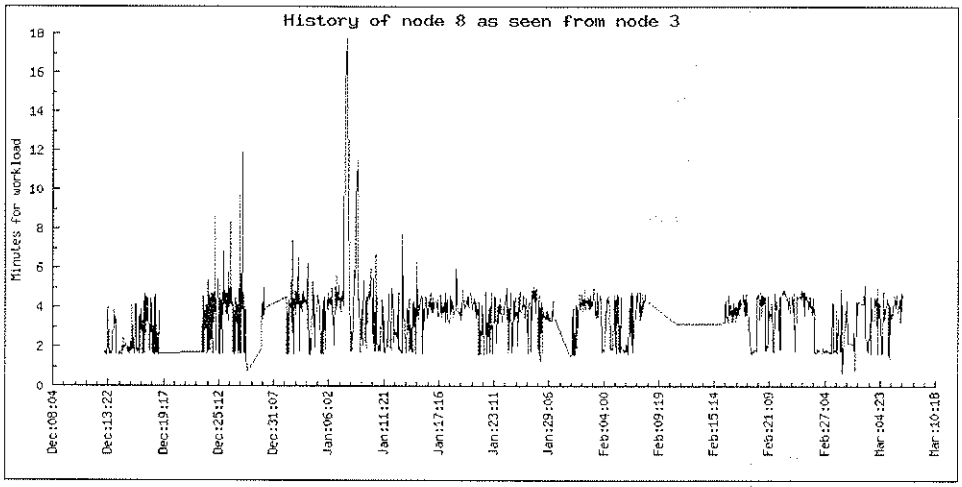


Figure 5.1: Long term remote resource performance vs. time (4 hour interval).

The results show that most nodes operate in a dual mode fashion. In one mode, they have a base performance level that is quite constant. In the second mode, they jump into an overload condition where their performance drops by a factor of 2 or more. We speculate that the overload

was probably due to high load experiments being run on the nodes by other researchers. This artifact can be observed in the blue line in Figure 5.1.

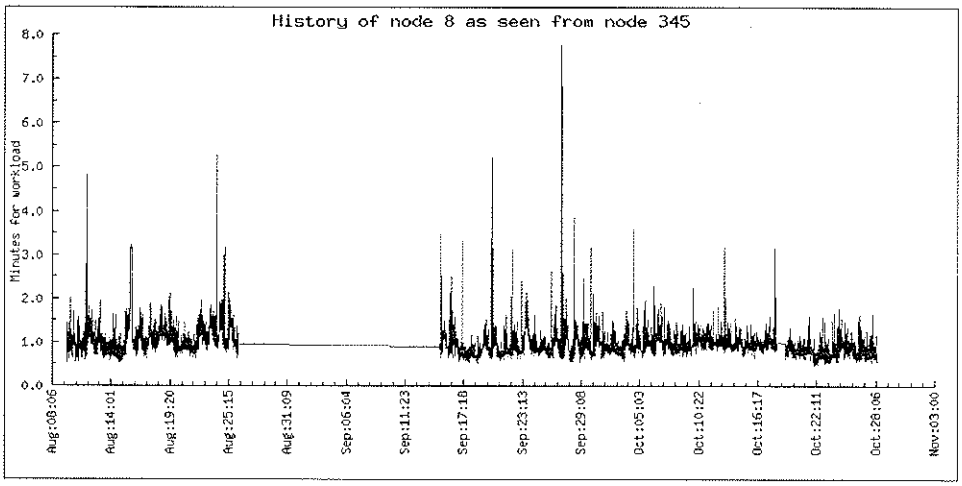


Figure 5.2: Short term remote resource performance vs. time (10 min. interval).

To be sure that this dual mode behavior was not an artifact of our selected 4 hour window, we ran a small test set of 5 nodes. These nodes tested one another on a 10 minute window with a reduced workload of 1.5MB. The results in shown in Figure 5.2 seem to have similar characteristics to the 4 hour window version.

From these results, we developed a simple moving average prediction algorithm shown by the red line in Figure 5.3. The algorithm selects a window of past data and takes the average and standard deviation in the window. It then predicts the future performance to be within x standard deviations of the mean. We defined a successful prediction to be one that fell between a lower bound, in which the performance was as good as the prediction, and an upper bound, in which the performance was within a factor n of the prediction. The lower bound was included to ensure that there wasn't an inordinate waste of resources. We then tested many values of the parameters n, x, and various window sizes, trying to minimize their values without dropping below a 95% successful prediction rate. We found a minimum in those values at n=3 for an upper bound value,

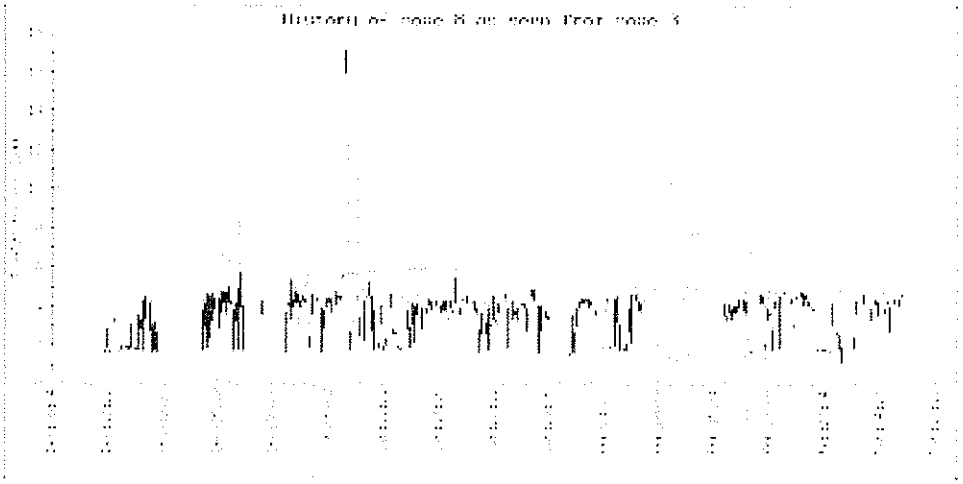


Figure 5.3: Remote resource performance and prediction vs. time (4 hour interval).

x=2.50 sigma and 6.35 days for the window size. These values gave a successful prediction 93.88% of the time as shown in Table 5.1. Table 5.1 shows the percentable of times of averaging algorithm produced false predictions as defined above, for each of the inputs.

5.1.3 Performance Prediction Experiments: Conclusions

From the last set of experiments, we found that with about one week of data our simple averaging algorithm works well for future performance predictions, for our application and dataset running on PlanetLab. One of the advantages of this algorithm is its simplicity. As the algorithm input is based on application specific plug-in performance evaluation, it should prove to be robust in light of new datasets and varying applications. More accurate prediction algorithms do exist but, as the averaging algorithms works well we decided to move on to other portions of the system rather then, explore improving our predictions with more sophisticated techniques.

		Deviation from the Mean (in sigmas)							
		1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50
Window Size in Days	3.15	8.30%	8.05%	7.91%	7.86%	7.91%	8.01%	8.14%	8.29%
	3.63	7.72%	7.46%	7.36%	7.29%	7.37%	7.48%	7.65%	7.80%
	4.17	6.82%	6.52%	6.44%	6.38%	6.47%	6.59%	6.79%	6.95%
	4.80	6.94%	6.64%	6.53%	6.45%	6.53%	6.66%	6.89%	7.11%
	5.52	7.01%	6.62%	6.45%	6.36%	6.42%	6.52%	6.76%	6.96%
	6.35	6.79%	6.36%	6.21%	6.11%	6.12%	6.27%	6.48%	6.73%
	7.30	7.12%	6.73%	6.58%	6.46%	6.54%	6.68%	6.93%	7.19%
	8.39	7.14%	6.73%	6.60%	6.57%	6.67%	6.85%	7.08%	7.34%
	9.65	7.25%	6.90%	6.85%	6.84%	6.96%	7.16%	7.47%	7.72%

Table 5.1: Percentage of false predictions over database of results for different window size and deviations from the mean.

5.2 Remote Resource Selection Algorithm

One of the key components a deployed system will need is the ability to balance new load as it is introduced into the system. We found some interesting phenomena in our first set of tests, such as the relationship between node locations and their view of resource performance. We sought to leverage these findings and the results from Chapter 5 two to produce a resource selection algorithm.

Similar work has been done in Grid research [51] [8] [32] [13]. These Grid based resource selection algorithms balance application specific design vs. complicated QoS agreements. We propose to use an application layer plug-in in place of a complicated QoS specification to drive our selection algorithm.

5.2.1 Remote Resource Selection Algorithm Testing

We returned to the database of results obtained in earlier in this chapter to develop our new algorithm. As the database contains a record of the performance of all of the peers, we have complete knowledge of almost all of the resources in each time slice. We say almost, as in the last 8 months we have had a few transitions in our PlanetLab management system and at any one time 10-20 nodes are offline. We generated a synthetic demand list for each node at various time slots to match the database data, areas of the data set where we did not have any data. The synthetic demand consisted of a series of plug-in demand values and durations. We generated demands for each of the anchor-point locations. We made sure that the demands avoided gaps in data. As we already know what the performance of a resource would have been, as seen from all of its peers, we can emulate the demand placement decision a node would have made. We tested various selection algorithms:

Load Based Based on examining the load on all of the resources and selecting the resource with the lowest load. This algorithm has been proposed by some research Content Delivery Networks (CDNs) such as Coral [23].

Range Based We then used an algorithm that selects the node nearest to itself. This is the algorithm that many commercial CDNs use, such as Akami [18]. Commercial CDNs do consider loading of data centers but, we ignore this as the dominant factor is range.

Uptime This algorithm is based on the length of time a resource has been online. In our case this means the number of time steps the resource has been returning data.

Local Learning Local learning uses the averaging algorithms from our Chapter 4 tests.

Random We also included a random selection algorithm as a litmus test to be compared against our other algorithms.

Global Learning Global learning is another implementation of an averaging algorithm, but is implemented as a shared object. We did not intend this algorithm to be implemented in a true deployment. Rather we expected it to act as an upper bound to our performance measures. As a shared object, the entire population of clients learn as a group by updating a shared data structure, rather than by acting independently.

Clairvoyance The Clairvoyance algorithm was another litmus test. It looked ahead into the database to see the performance of resources in the next time slot and used this clairvoyant information to make its placement decision.

5.2.2 Results of Resource Selection Algorithm Testing

Algorithm	Percent of Demand Satisfied
Global	76%
Clairvoyance	68%
Load	67%
Local	65%
Range	64%
Uptime	57%
Random	56%

Table 5.2: Traditional demand placement algorithms.

In Figure 5.2 one can observe that algorithms with some load balancing built-in did well. Clairvoyance didn't manage to do much better than load based, because load based was able to avoid systems that already had some demand placed on them. As expected, the random approach performed the worst, but not much worse than uptime. This can be attributed to the fact that the

random approach has some inherent load balancing built in. Our local learning technique from the previous experiments did not fare very well either. We expect this is because of its lack of load balancing, which would take no steps to avoid swamping the high performance nodes leaving the slower nodes unloaded. Figure 5.2 shows the average demand satisfied up to the current step in the plot. As the plot continues, values will level off.

5.2.3 Revisiting Resource Selection

Unsatisfied with the results obtained, we revisited the previous emulations. Here is a list of the algorithms we re-implemented in the redesigned emulator

- Random Selection
- Clairvoyance
- Range Based
- Local Learning

To further expand this set of emulations, we increased the number of anchor points by a factor of 6. We chose 6 because 6.3 nodes is the average number of PlanetLab nodes in an area. Often office or campus size networks have very small latency and high bandwidth. This distance is important as we make the assumption that communication delays on this metropolitan scale will be negligible. We leverage this low communication cost to allow nearby anchor points to form cooperative clusters. As nearby nodes tend to have a similar view of the network we reuse the same performance figures for each of the emulated anchor points. We explore self clustering in a later chapter and live rather than emulated data is used.

We then tried to leverage knowledge from the last set of experiments, together with the similar view nearby nodes have of the network. We then implemented a series of new algorithms:

Load Balancing on the Resource: In this algorithm, we enabled the resource itself to maintain a list of the performances experienced by the clients that use it. If a new client wants to make use of the resource, the resource can report what its average performance has been over the past N recordings.

Load Balancing on the Anchor Point Cluster: Each cluster had 6 clients. These were emulated clients. In the emulation, we assumed that the nodes had negligible communication cost among the group. This assumption is appropriate as we found that PlanetLab has on average 6.3 resources per city (50km range). In Chapter 9 we return to this assumption and find actual clusters further improving performance. For the moment the assumption is that all the nodes in a city size area have the same view of the global network.

We decided that if the clients cooperated, by telling one another that they were using a resource, they could have load balancing among the local cluster.

Shared Correction: In this algorithm, each resource maintains a "published" performance average over the last 6 days of jobs. The run times being averaged are returned to the resource from the client after the run is complete. The anchor points make use of this average as a first approximation of the resources future performance.

When the client returns the job run time to the resource it also sends a copy to the local anchor points. The anchor points then compare the run time of the last 6 days of jobs (their own local average) to the resource's "published" average. The local anchor points uses the ratio between the global "published" resource average and the "local average" to create a local "correction". The local "correction" is the ratio between the "published" prediction and the "local" prediction.

When scheduling new jobs the anchor points multiplies the "correction" with the global "published" prediction to find a performance prediction that has been corrected for its posi-

tion in the network. It then selects the lowest predicted runtime resource for the client.

When the anchor point selects a resource to deploy a job it notifies the other anchor points in the cluster. All the anchor points in the cluster adjust the expected performance of the resource is use by incrementing its "in use" variable by one. In subsequent scheduling, resources that have jobs running on it, will have its "published" performance divided by the number of jobs currently "in use" on that node.

Anchor points un-mark resources when they are notified by the client that the job is complete. Anchor points also periodically poll the resource to check to see in a client's job is still running. If it stopped or the client becomes unresponsive (ping alive) the anchor point will cancel the job and decrement the resources "jobs" counter, to free it up for new jobs.

By making use of local load balancing, the algorithm avoids over loading nodes. Some interference may occur from other anchor points, but this would soon be reflected in the client performance numbers. On subsequent deployments the resource would not be as attractive.

It should be noted, that by making use of averages across many nodes globally, or locally, the influence a single malicious node, or set of nodes, can have is reduced. As well, a malicious node is easily detected by returning values far from the expected average.

Figure 5.4 shows the average demand satisfaction vs. emulation steps for the various algorithms. For each emulation step we plot the percentage of demands that have been satisfied up till the current step. The plot should, and does, level off to some fixed values as more accumulates. Neither the traditional algorithm nor our new simple algorithms perform very well. However, our shared correction algorithm performed even better than the clairvoyant algorithm. Its true strength comes from leveraging the large client base to average the resource's global performance while a cooperative cluster learns a local view. This algorithm forms a global cooperative network where all anchor points contribute to benchmarking the resource.

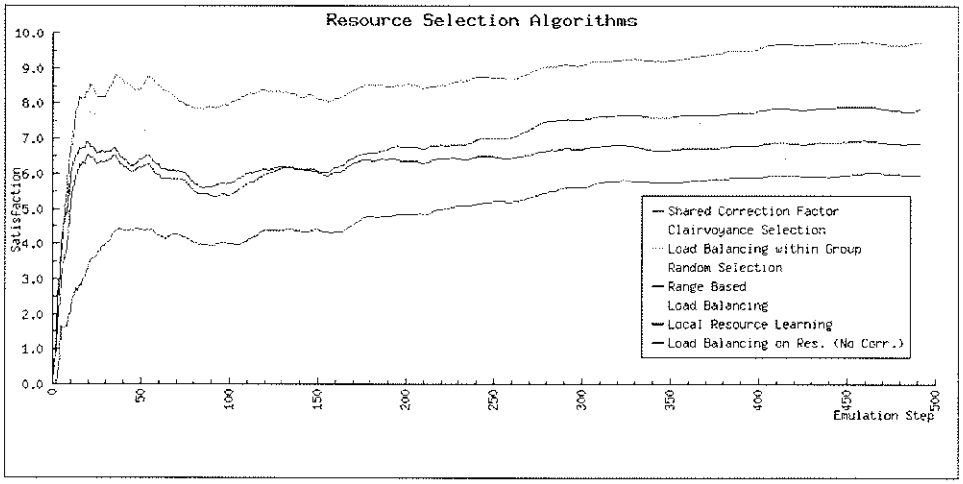


Figure 5.4: Cumulative percentage of demand satisfied vs. time step.

The Shared Correction algorithm achieves top performance by taking into account the short term average loading of the resource. This published resource loading figure allows all anchor points that make use of the resource to quickly learn the global loading of resources. At the same time a cooperating cluster of anchor points can work together to learn the "correction" factor from the resources' published value, allowing them to learn the nature of their position in the network. One key factor to making this cooperative cluster work is making sure the cluster is formed of nodes that share a similar network view. We show how these clusters are formed in Chapter 6 and 7.

5.3 Development

With a resource selection algorithm and performance prediction scheme in place we continue onto dynamically creating clusters of cooperative anchor points.

Chapter 6

Resource Clustering Study

We have observed in our emulations that clusters of anchor point neighbors can have advantages over anchor points acting alone. In our emulations, the clusters themselves were emulated at the same geographical location due to insufficient data. We are confident that clustering of resources shows good promise. To further explore this possibility we developed a series of tests to quickly establish if the clustering of resources is possible.

6.1 Ping Based Resource Clustering

In previous tests it was impossible to detect client clusters due to the small number of nodes being used. With only 40 nodes involved in the tests, cluster sizes would have consisted of only one or two nodes. In this test we made use of 400 PlanetLab nodes, giving 400 x 400 possible evaluations or just under 160,000 tests. With the large number of tests, each measurement needed to be small to reduce our total impact on the PlanetLab network. We chose to do a simple network ping. This did show some correlation in our original test of HTTP and transactional performance, as well as being simple to implement. We acknowledge that the ping test will only provide simple latency information between nodes. As was shown in Chapter 4, this only has weak correlation

with application performance. These results show good promise and we intend to redeploy the tests with application layer tests. We ran the tests for 2 weeks, repeating the 160k tests every 4 hours.

6.2 Anchor Point Network Position

To cluster the anchor points in terms of the resources' latency, we defined each of the PlanetLab nodes involved in the test as both a resource and an anchor point. To define an anchor point's position in the network in terms of resource pings, we define the anchor point's position as a signature composed of a vector of ping times to the remote resources (see Equation 6.2). For the moment use the term signature despite that it contains all of the data collected. In Chapter 7 we should how a large portion of the data can be discarded forming a true "signature".

R_{1A} = Ping Results from Anchor Point 1 to Resource A

S_1 = Signature at Anchor Point 1

$$S_1 = [R_{1A}, R_{1B}, \ldots, R_{1N}]$$

Ping based anchor point location signature definition. (6.1)

6.3 Ping Based Anchor Point Clustering

To cluster the resulting anchor point locations, we developed a simple clustering algorithm. The goal of the clustering algorithm was to find anchor points that have similar views of the remote

resources. We wanted to be able to find these clusters of anchor points without all of the anchor points having a full set of resource data. To achieve these goals, we set two thresholds: 1) an individual resource measure match and 2) a sum of matches threshold (see Equation 2). The formula finds the number of remote resources with similar evaluation, and compares it to the number of common remote resources. Should a large enough percentage of the remote resource evaluations match the anchor points for a cluster. The clustered resources then use the average of their evaluations when comparing with other anchor points that may want to join.

Two Anchor Points will merge or cluster if $C(S_1, S_2)$ is True:

T1 and T2 are pre-seleceted threshold values

S_x Is the signature of anchor point S

R_{xy} Is the ping time to resource y as observed from anchor point x

$$C(S_1, S_2) : \left(\sum_{i=1}^{|S_1 \bigcap S_2|} R_{1i} \left\{ \begin{array}{ll} 1 & if(R_{1i}/R_{2i} < T1) \\ 0 & \text{Otherwise} \end{array} \right\} / \mid S_1 \bigcap S_2 \mid \right) > T2$$

Clustering algorithm. (6.2)

The implementation of the algorithm starts with each anchor point as an individual cluster. We then cycle through pairs of anchor points comparing signatures with our clustering algorithm (see Equation 6.3). If the result is True, we merge the two anchor points into a cluster. We continue until clusters stop merging. In the case where we are comparing two clusters, we take the average value for each of the component elements of the cluster. The pseudo code follows:

```
def:  merge(x[],y[]):
      for each i (return z[i] = (x[i] + y[i]) / 2)
clustering = True
while clustering:
    clustering = False
    for each S[]:
        for each S[]:
        if C(S[x],S[y]):
            clustering = True
            merge(S[x],S[y])
```

Figure 6.1 shows the geographical distribution of the clusters. The diamonds on the map indicate a cluster. The diamond clusters are centered on the average position of the cluster's members, (as can be observed, some fall in the middle of the ocean). The size of the cluster represents the relative number of anchor points involved.

From the clustering results we can see that resources' signatures tend to give good indication of geographical location by considering that many nodes tend to have similar views of the network. By setting our thresholds to .85 for two measures of a remote resource to be considered a match and .85 for the number of resource matches needed, we see that 167 clusters were generated from the 400 nodes used. That results in just over 2 nodes per cluster on average. PlanetLab requires two nodes per site to join the test bed, leading us to believe that these threshold values are well matched to the co-located PlanetLab nodes. As we relax the thresholds, the number of clusters drops, as can be observed in Figure 6.2, where we see that the number of clusters is 61, with large clusters forming in the North America, Europe and Asia. As the majority of the largest contributing centers are located in these regions, the results seem reasonable.

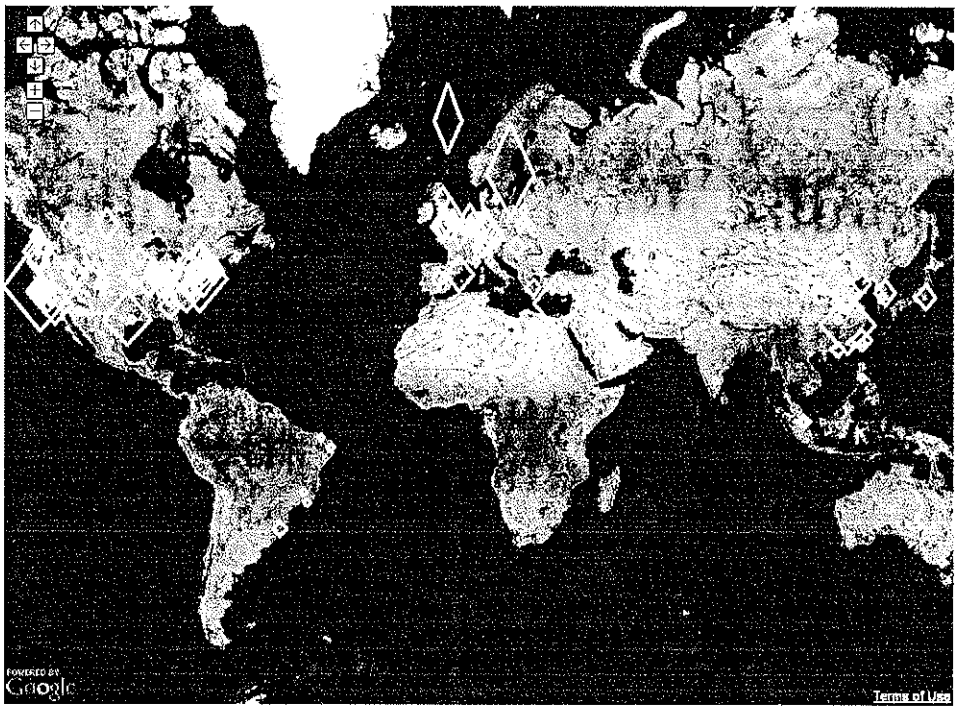


Figure 6.1: Global distribution of ping based neighbor clusters.

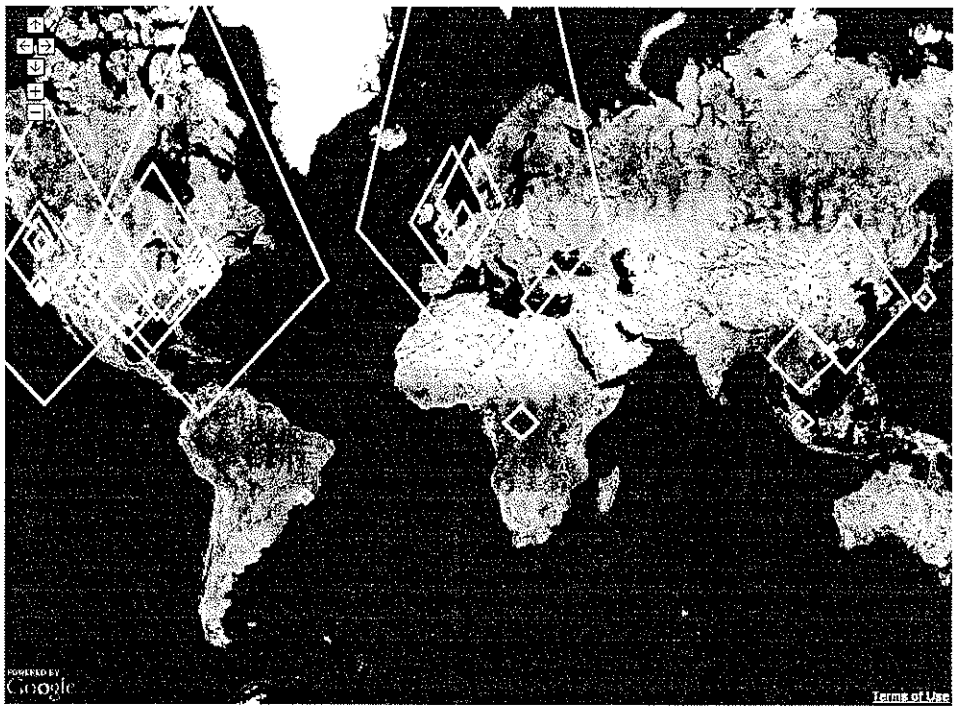


Figure 6.2: Global distribution of ping based neighbor clusters.

The results of ping based clustering show that finding clusters of anchor points based on resource performance measurement is possible. Unfortunately, basing clustering on pings alone may not be sufficient for applications that are computationally bounded. In these cases, ping times will be irrelevant. We should create and alternative method of clustering anchor points that can be better tuned to the deployed application.

Chapter 7

Resource Clustering

7.1 Probe Tests

The promising results based on ping time clustering inspired us to design a new set of experiments that would capture more than the simple latency between nodes. What we hoped to achieve was a clustering that could be customized to a variety of client applications. Simply using ping time may be appropriate for applications that make use of many small messages where latency is a consideration, but we would like our anchor points to be able to service computationally heavy applications as well. To address this short coming we undertook two sets of measurements.

Network Measurement

In the first set, we had each of the 400 PlanetLab nodes measure 50 pseudo randomly selected peer nodes and download five datasets using TCP. As can be observed in Table 7.1, the data sets ranged in size from 2KB to 20MB. We believe that this range should reflect many network intensive client applications demands, ranging from small transactions, such as instant messaging, to larger file transfer applications. Typical run times ranged from 9 to 83 seconds. We limited our maximum download size to 20MB as we expect the TCP sliding window to have stabilized to a steady transfer

Test Number	Download Size
Test 0	2KB
Test 1	20KB
Test 2	200KB
Test 3	2MB
Test 4	20MB

Table 7.1: Five TCP download test sizes.

rate. In general TCP transfers take some time to establish a constant rate of transfer. This is almost always much less the 20MB. Even in the rare case where it might not, there are few applications where a 20MB transfer would not be sufficient to characterize the network link.

7.1.1 Resource Measurement

Test Number	Number of Operations (x 1,000)
Test 5	0.5K Operations
Test 6	1K Operations
Test 7	5K Operations
Test 8	10K Operations
Test 9	50K Operations

Table 7.2: Five remote computation tests.

To further our network testing, we also deployed a set of five remote resource computing capacity tests. In these tests, we had the remote resource perform a number of integer multiplies to asses the resource's raw deliverable computing power. Each test consisted of 1,000 multiplies.

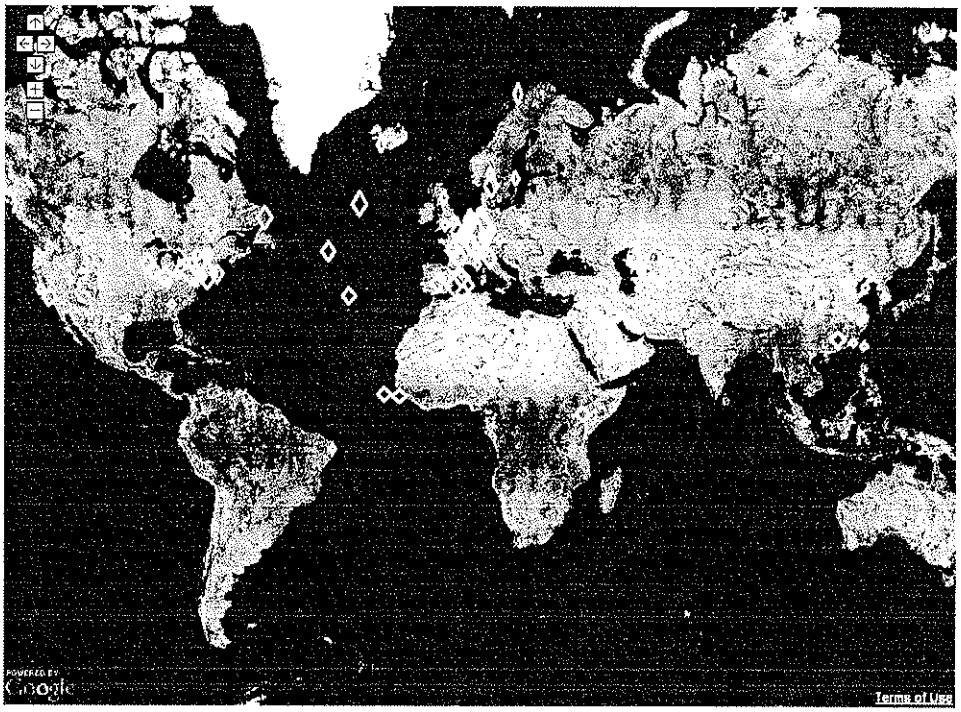
Again, we varied the loading on the resource through five tests ranging from 500 integer multiplies tests to 50,000 integer multiplies tests as can be observed in Table 7.2. typical run times ranged from 17 to 460 seconds. We limited our upper range to 50,000 multiplies when we found that the results stabilized.

7.1.2 Probe Based Clusters

By making use of the clustering algorithm used previously, we regenerated clusters using our probe data, with the results shown in Figures 7.1 and 7.2. As expected, the clusters do not form in the same locations when based on network download probes versus resource computational capacity. It should be noted that, the clusters (represented by diamonds) are placed in geographical average of the cluster leading to clusters appearing in the middle of the ocean.

It should be noted that our clustering algorithm works well without the full N x N test data we had in the previous chapter. As our clustering algorithm looks for the intersection of matching probe data, we needed to add a further constraint to the clustering algorithm. We added a minimum allowed size to the intersection of two signatures. If there were fewer than eight in the intersection, we did not subdivide further. We added this constraint to be sure that clusters were not being formed without sufficient data.

To evaluate the potential of the threshold values, we varied the X from 0 to 1, for each value of X we varied the value of Y from 0 to 1. In Figure 7.3, we see the resulting number of clusters when the X and Y thresholds are swept. It is interesting to see that the cluster count ranges from 10s of clusters to one cluster per node. This shows that the X and Y threshold values will allow us to adjust the cluster groups as needed. It is also interesting to note that the Y threshold sweep loses range as the X value approaches one. The X threshold is the threshold that two measures need in order to "match". Y is the number of "matches" needed for two anchor points to merge into one cluster. Figure 7.3 shows that, as we tighten the threshold of "matching" on one measure, all of the



Average Distance to Cluster Center: 342.845 Km
Number of Clusters: 105
Test Series: Compute Test #0
Threshold for individual resource agreement: 0.65
Threshhold for signature agreement: 0.65
Select Test Date: Tue, 28 Mar 2006 01:48:59 -0600

Figure 7.1: Geographical distribution of compute probe based neighbor clusters.

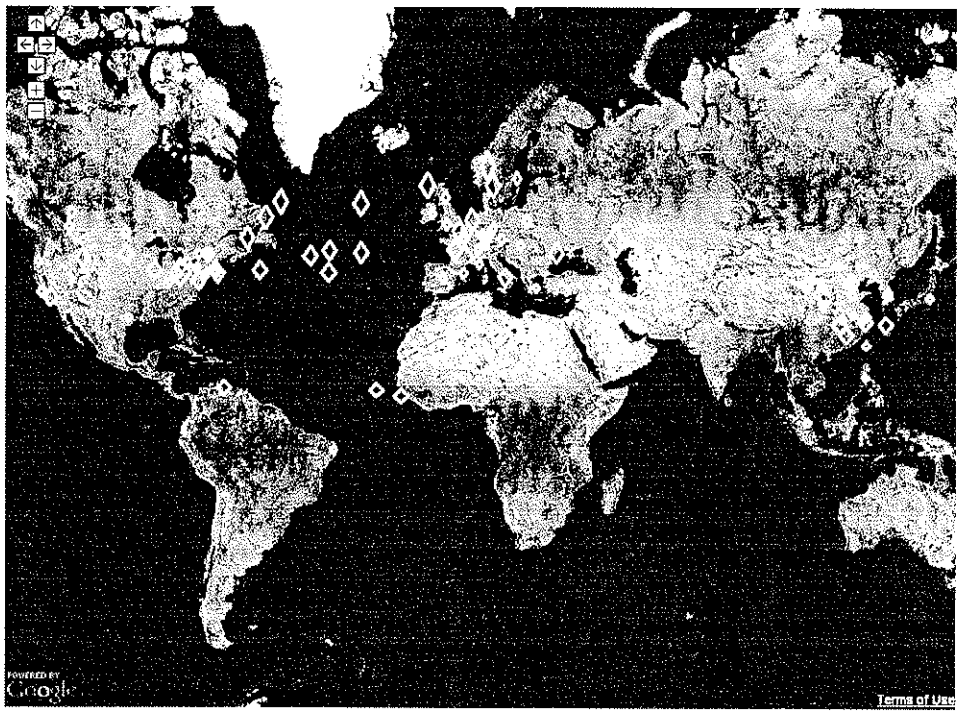


Figure 7.2: Geographical distribution of network probe based neighbor clusters.

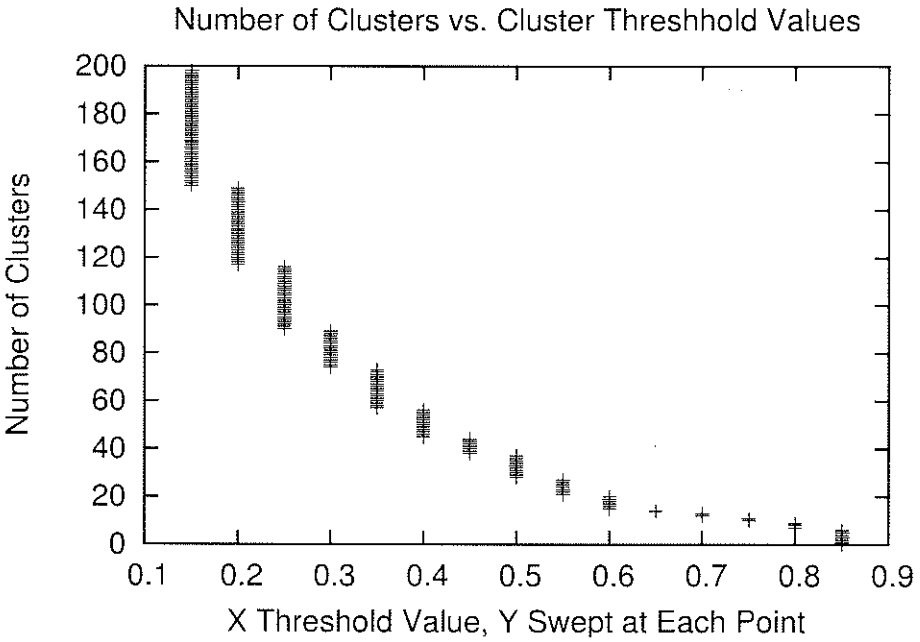


Figure 7.3: Number of clusters vs. X threshold, Y varied from 0 to 1 at each X value.

others must agree as the Y threshold loses its effect. We believe that this effect can be attributed to the strength of the X threshold. The fact that the Y threshold had little effect implies that all of the X values are all matching rendering the Y value irrelevant.

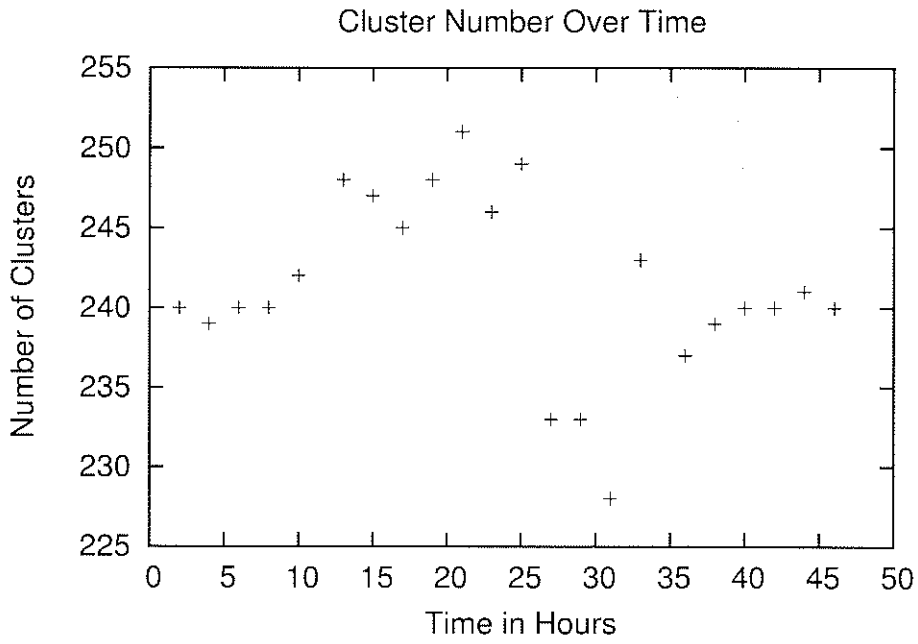


Figure 7.4: Number of clusters vs. number of hours.

In Figure 7.4, we have plotted the number of clusters over time given fixed values of X and Y threshold. As can be observed, there is very little variation in the number of clusters. All variation in this plot was caused by subset of large clusters swapping a small set of member resources. This could be addressed with a progressive clustering algorithm rather then one that regenerated the clustering at each time step. Overall the clusters are quite stable even in the face of fluctuation in network and resource loading. Our algorithms is effective because it averages many probes rather than over time as was the case in Chapter 5.

7.2 Deployment

Once our framework is deployed we envision a number of probes being run an a regular basis to ensure current data is available for each anchor point's view of the network. The probes used in a real deployment may be further tuned for the specific applications to be deployed. With a signature based view of the network, an anchor point will be able to contact peer anchor points looking for neighbor nodes to cluster with. When an appropriate node is found, they will merge.

7.3 Probe Based Clustering Results

We discovered that our clustering algorithm works well with probe data, even when faced with a reduced dataset. With this new probe data, anchor points will be able to customize their clustering to suit the application's network and resource needs. We suspect that this probe data can be further used to customize our framework to client applications. We explore this idea further in the next chapter.

Chapter 8

Probe to Application Matching

One of the drawbacks of basing performance predictions on performance histories is the need for historical data. As we saw previously, it takes on the order of 4-5 days for our prediction algorithm to achieve best results. It would be beneficial to have a priming system that would allow applications to be pre-benchmarked, allowing performance predictions before the application has filled the 4-5 day window. In addition, infrequent short lived jobs would benefit from an alternative means to generate performance predictions. From the previous chapter, we found a mechanism for clustering anchor points based on a set of ongoing probes. We propose that this on going probe information could be used to generate performance predictions of client applications.

8.1 Client Application Benchmarking

To further leverage the probe data that we are accumulating, we benchmarked the client applications in terms of probe data. To do this, we will compare the performance of an application to the current probe data for a subset of the resources. From the comparison of probe data to client application performance, we distilled a performance ratio of probe values to application performance. With this ratio, we can then further predict application performance using probe data from other

resources.

To generate these ratios, we deploy the application onto a subset of resources to measure its performance over a short time period. With the resulting performance data, we generate a vector of weights that correspond to the relative performance between the application and the particular probe from that resource (see Equation 8.1). We can then use these weights with probe data from other resources to give a performance prediction for the application on that resource.

$$WV = WeightVector$$

Perf(pn) = Performance of Probe n

Perf(a) = Performance of Application a

$$WV = [Perf(p1)/Perf(a), Perf(p2)/Perf(a), \ldots, Perf(pn)/Perf(a)]$$

Weigh vector calculation. (8.1)

To evaluate this technique, we deployed a series of application tests. For these tests, we made made use of the workload from Chapter 4. For each of the 400 PlanetLab nodes, we pseudo randomly selected 10 remote resources and deployed our workload. Once the workload was complete, we generated a weight vector by comparing the probe data with the workload runtime. With this weight vector, we used probe data from the 40 other resources to generate performance predictions. We then deployed our workload on the remaining resources and compared the results to the prediction.The pseudo code for the signature generation algorithm is as follows:

```
for p in probe.keys():
    sumSig[p] += AppPerf / probeData[p]
```

```
cntSig ++
appSig = []
for s to cntSig:
     appSig[s] = sumSig[s] / cntSig
print appSig
```

With these weight vectors, we can use existing probe data to generate a prediction for application performance on a remote host. This prediction can be used until enough live performance data accumulates to generate a performance prediction as developed before. The pseudo code for the prediction algorithm is given below:

```
sum =0
for p in probe.keys():
     sum += probeData[p] * weights[p]
prediction = sum / p
print prediction
```

As we see from the pseudo code, generating a prediction is a matter of using the weights and the latest probe data. In the actual source code, signatures with fewer then 5 remote locations in common were not included.

8.2 Probe to Application Matching Evaluation

To test the accuracy of our new probe based performance predictions, we pseudo randomly selected 10 remote hosts with probe data. We ran our Apache workload on each of these hosts and made use

of recent probe data to generate a weight vector. Making use of the weight vector, we generated performance predictions for 40 remote hosts that were not included in the generation of the weight vector.

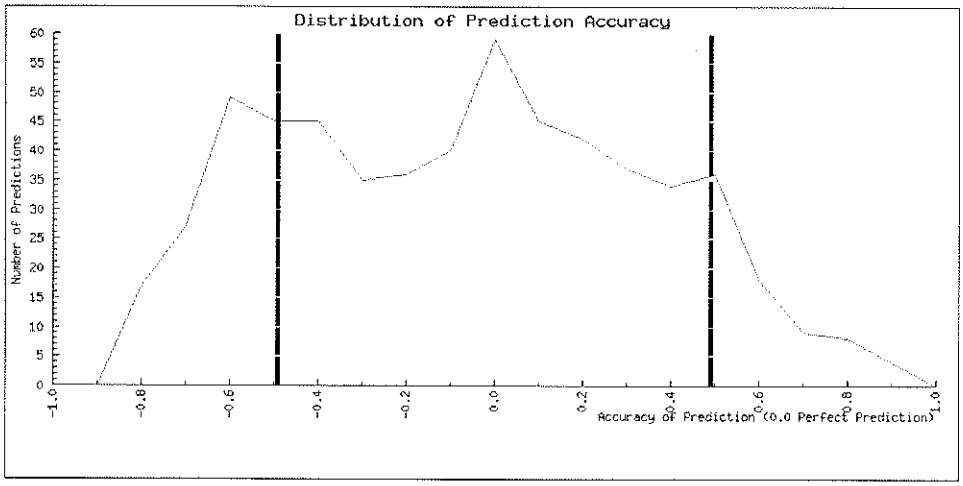


Figure 8.1: Distribution of performance predictions, frequency of measurement vs. error fraction.

Figure 8.1 plots the distribution of discrepancy between the predicted value and actual result. As can be observed, the bulk of the predictions fall within -0.5 and +0.5 or a factor of -2 and +2 (delimiting lines in black). In Chapter 5, we deemed a performance factor of two to be reasonable for a performance prediction.

8.3 Probe to Application Results

Through further study, the accuracy of weight vector performance predictions could be improved. At this point, we believe that our simple algorithm is sufficient to show that this technique could be used to generate performance predictions which compliment our "live" prediction algorithms developed in Chapter 5. We leave the optimization of this algorithm to proceed to an overall system evaluation.

Chapter 9

Framework Assessment

To do an overall assessment of our framework, a wide scale deployment would have been ideal. The deployment itself is not prohibitive, as PlanetLab has over 400 nodes distributed over the globe. Our testing and clustering algorithms using PlanetLab have shown good promise in this environment. Unfortunately, a large scale deployment for testing would require large scale loading of our framework. As our current contribution to PlanetLab is only two computers, it would be unfair to other PlanetLab researchers to place such a large scale load on the test bed.

9.1 Emulation Inputs

As an alternative to live deployment, we collected a series of probe and Apache Workload tests from the PlanetLab network. We then used this data to drive the emulator we developed previously in this thesis. As the emulator did not support probes and the programmatic clustering algorithm, we needed to make some enhancements to it. To truly test our large scale clustering, the size of the emulation had to be increased. We expanded the capacity of the emulator up to 400 nodes. Once our modifications were complete, we loaded our PlanetLab test data. To drive our emulation, we generated a synthetic anchor point demand workload. This workload pseudo randomly selected

N anchor points on which to start resource demands. For each of the N nodes a pseudo randomly generated level of demand was requested. For each demand, a lifetime for the demand was pseudo randomly generated.

It should be noted that during the framework evaluation we did not include the probe to application matching from the previous chapter. We believe that, without a live deployment, the value of this technique would be impossible to determine.

9.2 Emulation Algorithms

With our emulator and workload generated, we evaluated four different resource selection algorithms:

Random Anchor points randomly select the resources to deploy their load on.

Independent Nodes Anchor points work independently to choose which resource to deploy their loading on.

Simulated Cooperative Clusters As in our previous emulation, anchor points work cooperatively. The clustering of anchor points is imposed based on geographical position.

Probe Based Clustering In this algorithm, anchor points are allowed to probe the resources as described in Chapter 7. Making use of this probe data, the anchor points arrange themselves into clusters. These clusters leverage the fact that they have similar views of the network. The anchor point clusters work cooperatively using the same algorithms as the geographical clusters algorithm.

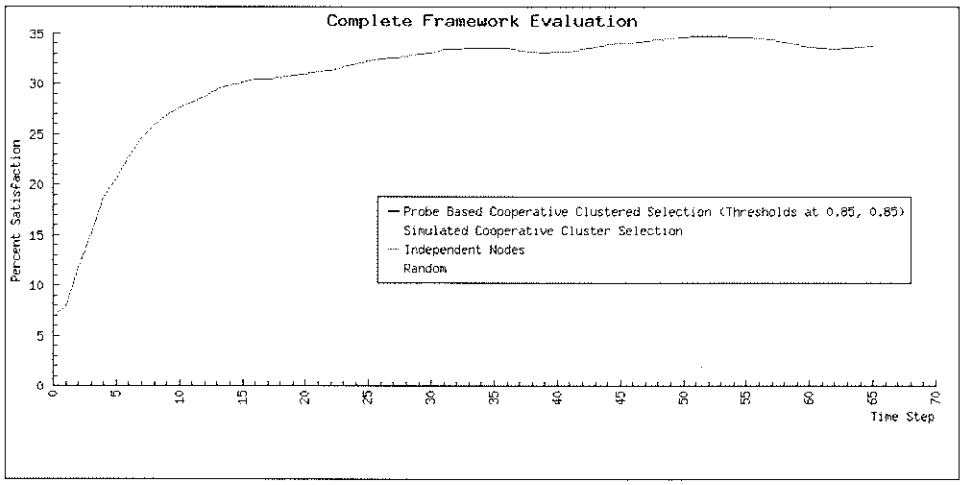


Figure 9.1: Comparison of probe based clustering and previous algorithms.

9.3 Emulation Results

We reused the random, independent nodes and simulated cooperative cluster algorithms from our previous emulation. We repeated the algorithms to ensure that our changes to the emulator and workload had not affected the results. In Figure 9.1, we can see that, as with the smaller scale emulation done before, the random, independent nodes and simulated cluster had similar performance as in the earlier experiments. The probe based clustering showed even better results. We believe this is because our programmatic clustering can better form cooperative groups. These cooperative groups are able to better predict the performance of a peer node, as they share a true shared view of the network. In the emulation, the nodes suffer when clustered by geographical distance due to their differing views of network resources. When forced into artificial clusters, the anchor points may not have the same network view they would have as clustered by our probe based clustering. The discrepancy between these two techniques would mean the clusters would not allow members of a cluster to learn an accurate correction for remote resources. Without accurate correction values, the anchor points would have trouble predicting the performance of remote resources. The

figure shows results as "percent satisfaction", which is the average satisfaction of the emulation. For example, 35% satisfaction implies that 35% of all the demands placed on the network have been satisfied.

The emulation was driven by a demand trace file. The synthetic demand consisted of a series of plug-in demand values and durations. We generated demands for each of the anchor-point locations. Each anchor-point tried to satisfy the demand placed on it by deploying simulated applications on resources. The performance of the application was taken from the existing database of performance traces. The demand trace was designed to over tax the system. As can be observed in Figure 9.1, we see that overall demand satisfaction never exceeds 35%. We could of course re-generate a demand trace such that all of the algorithms performed better.

9.4 Scalability

Scalability, as a property of systems, is generally difficult to define [25][19]. On the whole, it relates to the ability of a system to continue to function under the stress of increased loading. Some of the dimensions that are often measured include:

- Load scalability - As a system sees more load, the system performance gracefully degrades.
- System Size - As a system grows in resources, it continues to scale, this usually relates to not having any single bottleneck
- Administrative scalability - As the size of the system grows, more and more organizations can join in order to keep the number of systems component a single management entity must manage reasonable.

In the following section we examine each of these scalability factors for existing RMS systems:

9.4.1 Grid RMSs

[21], [15] and [22]

Grid RMSs [15] tend to have tight controls over the executing jobs as they are optimized for throughput over scalability. They also tend to tightly control the contributed resources not allowing the resources to be used for anything but grid jobs. Because of these tight controls, grids tend to be very scalable.

Load Scalability

In a grid RMS are generally load scalable as they have a strict admission scheme that allows them to control the system load. With such a mechanism they can deny access to incoming jobs curtailing an overload condition. Even in the situation where jobs are allowed to enter the system, they lead only to graceful performance degradation, as the job placement algorithm will try to maximize overall system performance [21].

System Size

Grids have proven to be very scalable. They have been used to manage sets thousands of computers. Unfortunately, when general purpose applications are deployed over a wide area network, the latency and reduced WAN bandwidth can limit performance. This is not a problem with the grid itself, but rather the application's scalability. Grids today do not have a good mechanism for modeling resource pools over WANs that can be easily adopted by application programmers. Grid applications are custom written using grid toolkits [22]

Administrative Size

Grid RMS developers have spent a large amount of time developing the systems needed to share grid pools among administrative domains. Grids maintain the ability for local grid pools to manage

their resources and still trade idle resources among themselves.

9.4.2 Volunteer RMSs

Volunteer systems limit the number of applications that can be deployed by only allowing communication to the central server. This affords volunteer systems many advantages and hurdles to scalability even through the capacity is available [6].

Load Scalability

One of the largest bottlenecks of volunteer systems is the central data center that the slave clients must contact for communication. This also severely limits the type of applications that can be deployed. On the other hand, as the central location does not do any computation, it can handle a relatively large amount of coordination.

System Size

Systems size scalability is not a problem for volunteer RMSs as the slave clients have no interprocess communication and the master slave communication is low. The total size of the network can become very large. Existing systems scale to 10s of thousands of clients contributing to a single application [5].

Administrative Size

Volunteer systems have poor contributer scalability as they have a single deployment point. Unfortunately, the application base is limited by the central communication architecture. Nearly every client is owned by a different party. From the client perspective they are very scalable.

9.4.3 P2P RMSs

P2P system are built for scalability rather than over performance. They also tend to focus on single applications. This allows them to optimize their operation for the single application chosen [43].

Load Scalability

Loading on P2P networks tends to scale very well as the load is designed to be distributed over the network as a whole. The loading is often in a tit-for-tat fashion. This means that users that place large amounts on the network need to contribute large resources. This fair sharing forces good scalability into the network but limits the applications that can be deployed.

System Size

The size of P2P networks is usually their strength. Most P2P networks assume little resources at each of the contributing nodes and attempt to build resources by leveraging a large resource pool.

Administrative Size

As each of the nodes is owned by a different user, P2P networks are the most scalable RMS today. They often have 10s of thousands of users, who each perform the necessary administration.

9.4.4 CDN RMSs

CDN RMSs are purpose built for scaling up HTTP traffic. They are designed to address the scalability issues with today's web servers [23].

Load Scalability

CDNs, like volunteer systems have a single distribution point at the origin server. This usually causes no problem as the redirection is a simple operation. In addition, the CDN pool is usually

very large, allowing load balancing among the CDN web sites.

System Size

Modern CDNs serve 100s of web sites with 1000s of resources geographically distributed around the globe. These networks are limited as they only serve a single application.

Administrative Size

CDNs avoid the issue of administrative scalability as they are usually owned and operated by a single entity. The client web servers do not handle very much of the system load and are, of course, owned by another party.

9.4.5 The Developed Framework

Our framework attempts to generalize these system into a novel system that scales in all of the dimensions just discussed. As we saw, each of the preceding RMS suffer in at least one dimension of scalability or another.

Load Scalability

Our network is designed to be load scalable. The total client loading can be equally distributed over the total resource base through the anchor points. Our RMS has another advantage in that, by our on-line benchmarking technique, clients gain a corrected view of the resources available, unlike any of the existing RMSs. The limiting factor is the loading placed on the anchor points. In the degenerate case, all of the load could be placed onto one anchor point. In this case, the anchor point would need to replicate itself onto some of the free resources in the pool.

System Size

As our RMS takes into account the network in its placement algorithm, it can hide latency considerations from applications. This isolation feature is one of the major difficulties in grid RMSs. The distance isolation feature also allows for system size scalability that no other RMS can match. The implicit reduced value of distant resources allows for system scalability that could potentially scale globally.

Administrative Size

As each of the resources and clients in our RMS can be owned by a different person, the administrative scalability is as good as P2P networks. The only limiting factor is the need for a starting, or root, anchor point that can replicate itself within the resource pool. As the anchor point pool can run on any of the contributed resources this should not be a limiting factor.

Scalability is an important measure for any Resource Management System (RMS). As all of our testing and results are products of the PlanetLab test bed, we believe that our framework has been grounded in reality. Further, our algorithms have been designed to be decentralized, thereby mitigating any overloading of a single component. Good support for this is provided by our emulator. In our first round of emulations, we started by emulating 40 nodes in the network. In our last set, we had 400 nodes where our algorithms continued to operate correctly. We believe that the system can handle any number of nodes participating. Some of the limiting factors will be:

- As the application set grows, we may need to tune the probes that are used for clustering. As new applications are added, we may find that our probes do not capture the demands that applications are putting on the resources. Adding new probes at a later time is quite simple in our framework and need not be implemented by all nodes at the same time.

- There are two threshold parameters used to adjust the size of clusters that are generated from the probe data. As the size of the network grows we may need to limit how many

nodes cooperate in a neighborhood. These parameters may need to be adjusted to limit neighborhood sizes.

- We have yet to design a discovery and dissemination scheme. The distribution of resource and anchor point lists will need to be done in a scalable way. One advantage this system has is its robustness in dealing with non-current resource and anchor list data. The propagation of erroneous resource or anchor point information will not seriously affect total system performance.

- We will also need to further tune our algorithms and probes as new applications with different network and resource demands are added.

If these four shortcomings are addressed in a framework deployment, we believe the system to be very scalable to a global size. One major advantage we have over existing RMS is basing our performance predictions on past application performance. By using this technique, our evaluation of resources includes all of the network and resource limitations. Leveraging this data, we can use the resources for anchor-point placement, allowing them to be deployed as needed and making the RMS self healing.

9.5 Robustness

Robustness is an important quality of a Resource Management System (RMS). As our system is designed to be distributed, it is by nature globally robust. The loss of large segments of the network or segmentations will not have a large impact, other than the loss of capacity. Berg [?] defines robustness as "a measure of the degree of a system which is in the operable and committable state at the start of mission when the mission is called for at an unknown random point in time". In effect this is directly related to the "uptime" of a system.

In our experiments we had intended to do a more rigorous examination of robustness through a study of uptime and availability on PlanetLab. Unfortunately, during our PlanetLab tests we had a number of difficulties maintaining accurate data. Some of the down time we captured can be attributed the PlanetLab down time. But the large majority of the down time we saw was due to programming errors, server migration problems or configuration errors on our part. As the majority of the down time was due to our own efforts, we decided to simply do a single point of failure analysis, rather then a study of our own induced reliability faults.

If we examine some of the existing RMS we see that some are not robust due to existence of a single points of failure within their design.

- Grid RMS - Grids suffer from loss of capacity due to a WAN failure also a possible loss of data. This is due to the type of applications that run on a grid rather than the RMS itself.
- Volunteer RMS- Volunteer RMSs Suffer from a major issue of having a centralized point of failure; the master data center. A loss of the master data center renders all processing and data offline.
- P2P RMS- P2P RMSs only suffer from a loss of capacity and data if un-replicated in the network, an unusual event.
- CDN RMS- CDNs Suffer from a loss of access to web data if the central server goes offline. Usually, the CDN will used cached data.

From a design perspective, our RMS does not suffer from any central point of failure like CDNs or Volunteer RMS's. There are still some points that can fail in our RMS, but they will not result in a catastrophic systems failure. Such possible failures points include:

- Neighbor Failure: Should an anchor point neighbor fail, it would be quickly discovered and disseminated to the other neighbors. The moment any neighbor receives data concerning

one of its application's performance, it needs to share it with all its neighbors. Had one or more of the neighbors failed, they would be removed from the neighborhood. The neighbor could, of course, re-join following its recovery through the normal clustering mechanism.

- Resource Failure: As resources are constantly being evaluated by application plug-ins and probing, any reduction in performance, whether from overload or failure would immediately be noticed. The anchor points would need to start migrating applications to free resources. In the event of a catastrophic failure, any affected anchor point would be obligated to start large scale trimming of its load by reducing service to its clients.

- Anchor Point Failure: Should an anchor point fail, the client plug-in running on the client would notice immediately as it is feeding performance results to the anchor point. As the other resources would still be operating and supplying the data, the interruption in service would not be immediate. Once the anchor point failure is detected, the application plug-in can begin to move its resource demand to another anchor point in the neighborhood or elsewhere.

The only remaining item of the framework that can fail is the wide area network itself. Should a client lose access to the network, there is nothing that can be done. We see that, by making use of on-line performance data, the system is relatively insensitive to failures.

9.6 Security and Trust

One important area of consideration for any Resource Management System (RMS) is security. In systems where not all components are owned by the same party, trust can be a problem as well. Before our framework could be used in a commercial deployment, these issues would need to be addressed. At the moment, we have put them aside for future study.

In defense of this discussion security can be easily added though encryption of channels and secure key exchange, PlanetLab already has these facilities enabled. Trust can be built into the feed back mechanism of the anchor points. As the anchor points maintain an average performance number for an area it is easy to check the validity of returned results. Should clients return false data on a regular basis they could have a "trust" level reduced and weighted accordingly.

9.7 RMS System Comparison

Some aspects of our framework are shared with existing Resource Management Systems (RMS). We compare our framework to existing systems, Table 9.1.

Attribute	Grid	Volunteers	P2P	CDNs	Framework
Donated Resources	no	yes	yes	no	yes
Number of Applications	many	many	one	one	many
Dedicated Resources	yes	no	no	shared with QoS	yes
Predictable Performance	yes	no	no	no wide area affects	Some QoS
Level of Distribution	partial	centralized	fully	partial	fully

Table 9.1: Resource management system attributes comparison.

We observe that our framework can make use of volunteered resource like p2p and volunteer systems. Our framework can also make use of dedicated resources similar to cluster computing or grid systems. Our framework is fully distributed and self-organizing like p2p networks thereby making the system more robust. We also have some measure of QoS through performance prediction, similar to grid systems. We can also support many applications unlike p2p or CDNs.

9.8 Summary

Making use of our probe data to generate anchor point clusters, rather than geographical clustering, we found improved results from our previous tests where anchor point clusters were simulated. This was due to the probe based clusters being a much better measure of the shared network path than trying to infer clusters from geographical topology. From these results, we can see that probe based clustering works well as a means to self organize anchor points into cooperative clusters. Other factors that we considered in the design are scalability, robustness and security. The scalability and robustness of the framework are integral components of the design of the algorithms we used. Security and trust still need to be addressed in the future work.

Chapter 10

Conclusion and Contributions

10.1 Overview

We have developed a novel framework for the management of a network of shared resources. The framework uses a set of anchor points, which probe remote resources to establish resource clusters among anchor points. The probe data also, provides baseline resource performance. The anchor points cooperatively manage the loading of resources to ensure that client performance is maintained.

10.2 Thesis Contributions

This thesis describes an empirical application performance measurement based RMS. The framework itself, probe based clustering and performance prediction algorithms are novel and have application in other domains and systems.

10.3 RMS Framework

Similar frameworks that use anchor points, or "landmarks", exists in experimental Label Switched Paths (LSP) [34] and sensor networks [31]. In these situations the landmarks are used either as fixed WAN locations or for host positioning. Our framework places these node on the edge of the network and perform significantly different and larger task then traditional "landmarks". This is one of the reasons we describe our "landmarks" as anchor points. Our framework itself is a novel development. Rather than have fixed points on the network as is the case many RMS system to act as landmarks, we propose that our anchor-points can be moved off the WAN backbone and into a LAN environment. The anchor points act as ambassadors to the local network learning the site specific attributes of the network position and the effect of the site on application performance. Our framework tries to be application independent, although for the purposes of developing our RMS we used database and web applications. The RMS itself is designed to be adaptable to any online application that could make use of a large, wide area pool of resources. We believe these aspects make the framework a valuable contribution that could be deployed or leveraged by grids, CDNs, P2P or volunteer RMSs.

10.4 Application Performance Measurements

Application performance measurement has been previously used in the modeling of application performance in grid systems. The use of application performance has not been used to evaluate the end to end performance of a general purpose RMS. One reason is by only looking at the application performance there is no information for the developer to find bottlenecks. Without bottleneck information there is no clear indication of where to make network improvements. Application performance measurement is used in P2P networks for file contribution assessment. The use of application performance in a RMS has not been explored before this work.

10.5 Performance Predictions

We discovered that performance predictions can be made at the application using a simple moving average. We do not consider the moving average a major contribution but rather a good indication of the strength of application performance measurement as a technique. Even with a naive filtering algorithm our emulations show good promise.

10.6 Resource Selection Algorithm

We developed a novel distributed cooperative algorithm for resource selection. This algorithm makes use of simple application performance averaging for resource selection. The combining of global performance measurement with local network correction is a novel aspect of our work. The algorithm is very simple and is designed to be application independent. We believe that there is still room for improvement in the prediction algorithm leveraged by the resource selector. Despite the simple moving average prediction, our two part cooperative resource selection algorithm performs well.

10.7 Anchor Point Clustering

One major contribution of this thesis is our discovery of clusters of resources in the network that share similar views of the remote resource world. We developed an algorithm that can cluster resources based on their views of the remote resource world. By clustering together resources with similar views we found that they can act as a cooperative unit. This knowledge could also be exploited to determine the number of anchor points a network needs. This clustering view can also be used as an application level technique for determining network topology of networks where no lower level data is available.

10.8 Application Profiling

Finally, we found that application performance benchmarks can be found based on a set of generic application probes. The actual benchmarks could be improved with further study, but even with the simple measures we studied, web performance could be found with probe data without needing to deploy the application. This style of application benchmark warrants further study, but based on our initial simple tests, this technique shows promise.

10.9 Summary

Our empirical measurements of application performance based on probing and clustering has not been previously explored as a mechanism for performance prediction or as a basis for an Resource Management Systems (RMS). We have also developed a fully distributed self organizing framework. Clustering resources based on the performance observed at remote locations has not been previously exploited. This framework also incorporated existing technologies to build a skeletal framework that is application independent. Due to its application independence it we believe that our framework can be applied to any number of applications including, remote computation such as weather prediction [30], computer graphics rendering [17] or other distributed applications such as web services.

Chapter 11

Directions for Future Work

The framework developed is very skeletal and application independent. To continue the work great gains could be made by restricting the application domain to one problem rather than continuing with a generic framework. As an example many of the applications that we would like to use in the framework, such as streaming media, would require an additional component for modeling server uptime and managing remote resource configuration time. We started with an initial study of server uptime but found our short deployment time did not give sufficient data to draw any conclusions. By restricting the application domain of the framework we could make additional assumptions that would draw out the application components and allow us to ignore the aspects that are irrelevant.

Another factor that is holding back our work is the large scale network needed to make further advances. Even with the 400 nodes of PlanetLab, our total allowed resource consumption is limited. To gain a true understanding of the universal applicability of our framework a much larger network with a wider application test set is needed. Again, focusing on a specific computational task would reduce this problem, allowing us to reduce the test network size and loading requirements for evaluation.

11.1 Concluding Remarks

Attacking a general problem such as an RMS will always be difficult. Despite years of exploration, no total solution has been found. RMS research will continue for many more years. We have developed a skeletal framework that shows good promise. We have also developed a number of novel techniques that could be applied to a full RMS or existing RMSs.

Bibliography

[1] Rosetta@home. In *http://boinc.bakerlab.org/rosetta/*, July 2006.

[2] Seasonal attribution project. In *http://attribution.cpdn.org/*, July 2006.

[3] Seti@home. In *http://setiathome.berkeley.edu/*, July 2006.

[4] D. P. Anderson. Public computing: Reconnecting people to science. In *Proc. of The Conference on Shared Knowledge and the Web, Residencia de Estudiantes, pp. 90-93, Madrid, Spain,*, March 2004.

[5] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: An experiment in public-resource computing. In *Communications of the ACM November 2002/Vol. 45, No.11 pp. 56-61*, November 2002.

[6] D. P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *Proc. of Cluster Computing and the Grid, 2006. CCGRID 06, pp. 73-80. Sixth IEEE International Symposium*, May 06.

[7] D. P. Anderson, E. Korpela, and R. Walton. High-performance task distribution for volunteer computing. In *Proc. of The IEEE International Conference on e-Science and Grid Technologies 5-8 December 2005 pp. 8-12*, December 2005.

[8] D. Angulo, I. Foster, C. Liu, and L. Yang. Design and evaluation of a resource selection framework for grid applications. In *Proc. of The IEEE International Symposium on High Performance Distributed Computing (HPDC-11), pp. 63-71. Edinburgh, Scotland*, July 2002.

[9] M. Arlitt and C. Williamson. Web server workload characterization: The search for invariants. In *Proceedings of The 1996 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems, Article 110. Philadelphia, PA,*, May 1996.

[10] E. Bailey. Maximum rpm. In *http://www.redhat.com*, November 1999.

[11] D. Bernholdt and S. B. et al. The earth system grid: Supporting the next generation of climate modeling research. In *Proceedings of the IEEE, 93:3, March, 2005, pp. 485-495*, March 2005.

[12] R. G. Brown. Engineering a beowulf-style compute cluster. In *http://www.phy.duke.edu/~rgb/Beowulf/beowulf_book/beowulf_book/index.html*, May 2004.

[13] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp. 300-301*, New York, NY, USA, 2003. ACM Press.

[14] L. Cherkasova, Y. Fu, and W. T. A. Vahdat. Measuring and characterizing end-to-end internet service performance. In *Journal ACM/IEEE Transactions on Internet Technology, (TOIT)*, November 2003.

[15] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets, 1999.

[16] D. Chua, E. D. Kolaczyk, and M. Crovella. A statistical framework for efficient monitoring of end-to-end network properties. *SIGMETRICS Perform. Eval. Rev., pp. 390-391*, 33(1), 2005.

[17] R. Cook, P. T., and C. L. Distributed ray tracing. In *Computer Graphics (Proceedings of SIGGRAPH 1984) 18 (3), pp. 137*, 1984.

[18] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. In *IEEE Internet Computing. pp. 50-58*, September 2002.

[19] L. Duboc, D. S. Rosenblum, and T. Wicks. Doctoral symposium: presentations: A framework for modelling and analysis of software systems scalability. In *Proc.of The 28th international conference on Software engineering ICSE '06, ISBN 1-59593-375-1, pp. 949-952*, May 2006.

[20] J. Flinn, D. Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proc. of Hot Topics on Operating Systems (HotOS-VIII), pp. 61-66, Schloss Elmau, Germany*, May 2001.

[21] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing, pp. 115-128*, 11(2), June 1997.

[22] I. Foster and C. Kesselman. Computational grids, isbn 1-55860-475-8. pages 15–51, 1999.

[23] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04), pp. 239-252. San Francisco, CA*, March 04.

[24] F. Gaglinardi and F. Grey. Old world, new grid. In *IEEE Spectrum Vol 43 Issue 7 pp. 28-33*, July 2006.

[25] M. D. Hill. What is scalability? In *ACM SIGARCH Computer Architecture News, December 1990, Volume 18 Issue 4,pp. 18-21, (ISSN 0163-5964)*, May 1990.

[26] K. Keahey, T. Fredian, Q. Peng, D. P. Schissel, M. Thompson, I. Foster, M. Greenwald, and D. McCune. Computational grids in action: the national fusion collaboratory. *Future Gener. Comput. Syst., pp. 1005-1015*, 18(8), 2002.

[27] S.-J. Lee, P. Sharma, S. Banerjee, S. Basu, and R. Fonseca. Measuring bandwidth between planetlab nodes. In *Proc. of (PAM) Passive and Active Measurement, pp. 292-305*, 2005.

[28] D. Lu, Y. Qiao, P. Dinda, and F. Bustamante. Characterizing and predicting tcp throughput on the wide area network. In *Proc. of 25th International Conference on Distributed Computing (ICDCS 2005), pp. 414-424*, 2005.

[29] M. Maheswaran, B. Maniymaran, S. Asaduzzaman, and A. Mitra. Towards a quality of service aware public computing utility. In *1st IEEE NCA Workshop on Adaptive Grid Computing (in the proceedings of 3rd IEEE Symposium on Network Computing), August 2004, Cambridge, Massachusetts, USA, pp 376-379*, August 2004.

[30] N. Massey, T. A. M. Allen, C. Christensen, D. Frame, D. Goodman, J. Kettleborough, A. Martin, S. Pascoe, and D. Stainforth. Data access and analysis with distributed federated data servers in climateprediction.net. In *Advances in Geosciences, 8, pp. 49-56*, 2006.

[31] T. Moscibroda, R. O'Dell, M. Wattenhofer, and R. Wattenhofer. Virtual coordinates for ad hoc and sensor networks. In *Proc. of The Joint Workshop on Foundations of Mobile Computing (DIALM-POMC'04), Philadelphia, Pennsylvania, pp. 8-16, USA. ACM 1-58113-921-7/04/0010*, October 04.

[32] D. S. Nikolopoulos and C. D. Polychronopoulos. Adaptive scheduling under memory pressure on multiprogrammed clusters. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 22-30*, Washington, DC, USA, 2002. IEEE Computer Society.

[33] K. Park and V. S. Pai. Deploying large file transfer on an http content distribution network. In *First Workshop on Real, Large, Distributed Systems (WORLDS '04), pp.134-142, San Francisco, CA*, December 2004.

[34] C. Pelsser. Using virtual coordinates in the establishment of inter-domain lsps. In *Proc. of The Conference on Future Networking Technologies (CoNEXT'05), pp. 59-64, Toulouse, France. ACM 1-59593-3/05/00010*, October 2005.

[35] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proc. of The First ACM Workshop on Hot Topics in Networking (HotNets) pp. 59-64*, October 2002.

[36] M. Ripeanu, A. Iamnitchi, and I. T. Foster. Cactus application: Performance predictions in grid environments. In *Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing, pp.807-816*, London, UK, 2001. Springer-Verlag.

[37] J. Rolia, L. Cherkasova, M. F. Arlitt, and A. Andrzejak. A capacity management service for resource pools. In *Proc of The Workshop on Software and Performance (WOSP'05): pp. 229-237*, July 2005.

[38] J. Rolia, L. Cherkasova, M. F. Arlitt, and V. Machiraju. Supporting application quality of service in shared resource pools. In *Commun. ACM 49(3): pp. 55-60 (2006)*, March 2006.

[39] A. J. Roy. *End-to-end quality of Service for High-End Applications*. PhD thesis, 2001. Adviser-Ian Foster.

[40] Y. Ruan and V. S. Pai. The origins of network server latency the myth of connection scheduling. In *Proc. of SIGMETRICS/Performance'04, pp. 694-698, New York, NY, USA.*, June 2004.

[41] M. Russell, G. Allen, G. Daues, I. Foster, E. Seidel, J. Novotny, J. Shalf, and G. von Laszewski. The astrophysics simulation collaboratory: A science portal enabling community software development, pp. 297-304. *Cluster Computing*, 5(3), 2002.

[42] F. D. Sacerdoti, S. Chandra, and K. Bhatia. Grid systems deployment management using rocks. In *Proc. of IEEE International Conference on Cluster Computing, pp. 337-345, San Diego.*, September 2004.

[43] K. P. Shanahan and M. J. Freedman. Locality prediction for oblivious clients, Febuary 2005.

[44] S. Sivasubramanian, B. van Halderen, and G. Pierre. Globule: a user-centric content delivery network. Poster in Proceedings at the 4th International System Administration and Network Engineering Conference, pp. 3-6, Sept. 2004. http://www.globule.org/publi/GUCCDN_sane_poster.html.

[45] W. Smith, I. Foster, and V. Taylor. Predicting application run times with historical information. *J. Parallel Distrib. Comput., pp. 1007-1016*, 64(9), 2004.

[46] S. Sobti, J. Lai, Y. Shao, N. Garg, C. Zhang, M. Zhang, F. Zheng, A. Krishnamurthy, and R. Wang. Network-embedded programmable storage and its applications. In *Proc. 3rd IFIP-TC6 Networking Conference (NETWORKING 2004) pp. 1143-1155*, May 2004.

[47] B. Spencer, T. Finholt, I. Foster, C. Kesselman, C. Beldica, J. Futrelle, S. Gullapalli, P. Hubbard, L. Liming, D. Marusiu, L. Pearlman, C. Severance, and G.Yang. Neesgrid:a distributed collaboratory for advanced earthquake engineering experiment and simulation. In *Proc. of The 13th World Conference on Earthquake Engineering, pp. 10-25, Vancover B.C., Canada*, August 2004.

[48] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality, ISBN 978-0470853191*. John Wiley and Sons Inc., December 2002.

[49] M. Z. J. unwen Lai, A. Krishnamurthy, L. Peterson, and R. Wang. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In *Proc. of The USENIX 2004 Annual Technical Conference. pp. 99-112*, June 2004.

[50] S. Vazhkudai and J. M. Schopf. Using disk throughput data in predictions of end-to-end grid data transfers. In *GRID '02: Proceedings of the Third International Workshop on Grid Computing, pp. 291-304*, London, UK, 2002. Springer-Verlag.

[51] S. Vazhkudai, S. Tuecke, and I. Foster. Replica selection in the globus data grid. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid, pp. 106*, Washington, DC, USA, 2001. IEEE Computer Society.

[52] S. Vidal. Yellow dog updater, modified. In *http://linux.duke.edu/projects/yum/*, December 2005.

[53] L. Wang, K. Park, R. Pang, V. S. Pai, and L. Peterson. Reliability and security in the codeen content distribution network. In *In Proceedings of the USENIX 2004 Annual Technical Conference, pp. 171-184, Boston, MA*, June 2004.

[54] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. In *Journal of Future Generation Computing Systems,Volume 15, Numbers 5-6, pp. 757-768*, October, 1999.

[55] L. Yang, J. M. Schopf, and I. Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *SC '03: Proceedings of the*

2003 ACM/IEEE conference on Supercomputing, pp. 31, Washington, DC, USA, 2003. IEEE Computer Society.

[56] H. Zhang, K. Keahey, and W. E. Allcock. Providing data transfer with qos as agreement-based service. In *Proc. of IEEE International Conference on Services Computing (SCC), pp. 344-353*, 2004.

[57] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of internet flow rates. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 309-322*, New York, NY, USA, 2002. ACM Press.

[58] N. Zhu, J. Chen, T. cker Chiueh, and D. Ellard. Tbbt: scalable and accurate trace replay for file server evaluation, pp. 392-393. *SIGMETRICS Perform. Eval. Rev.*, 33(1), 2005.